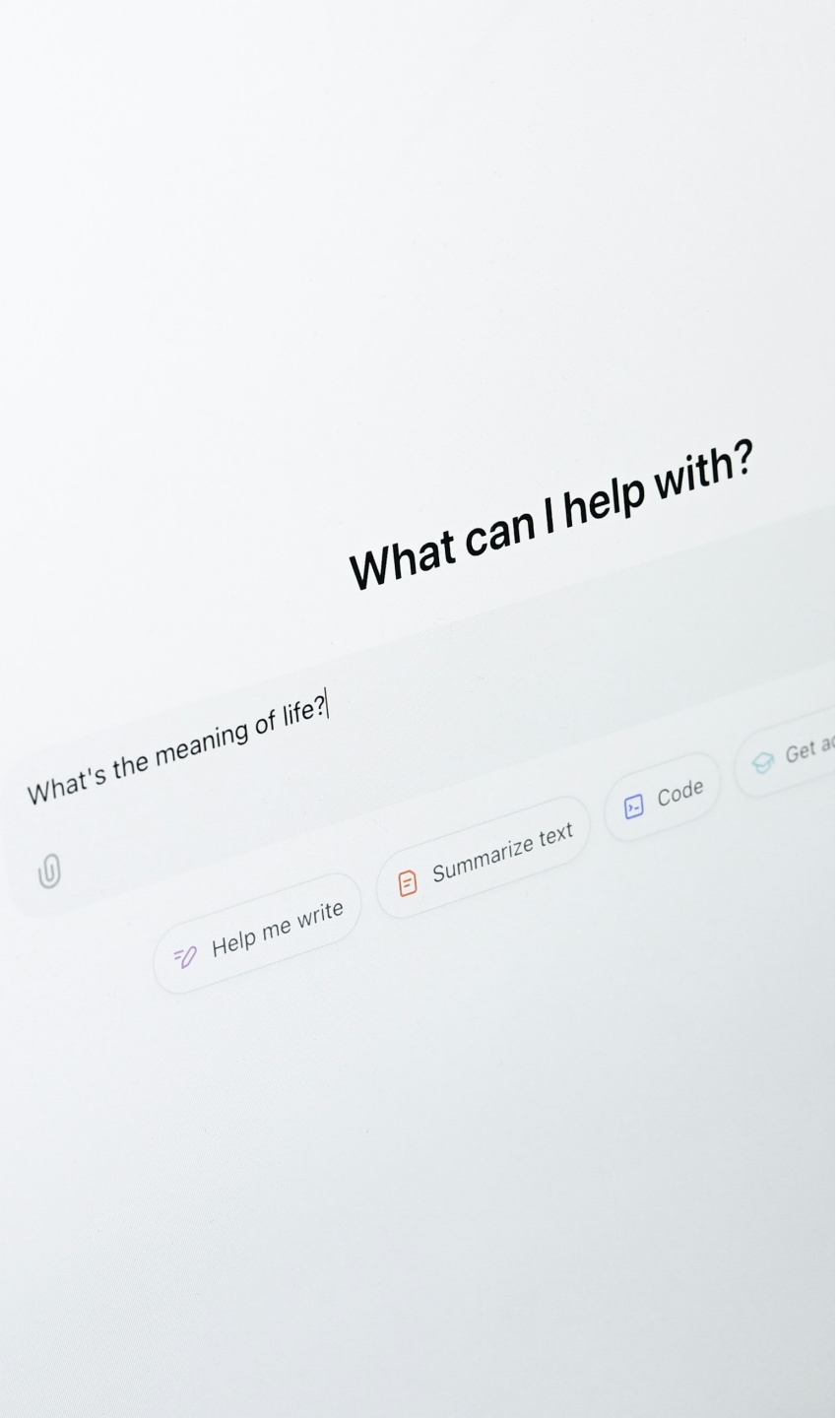


Introduction to Web Applications

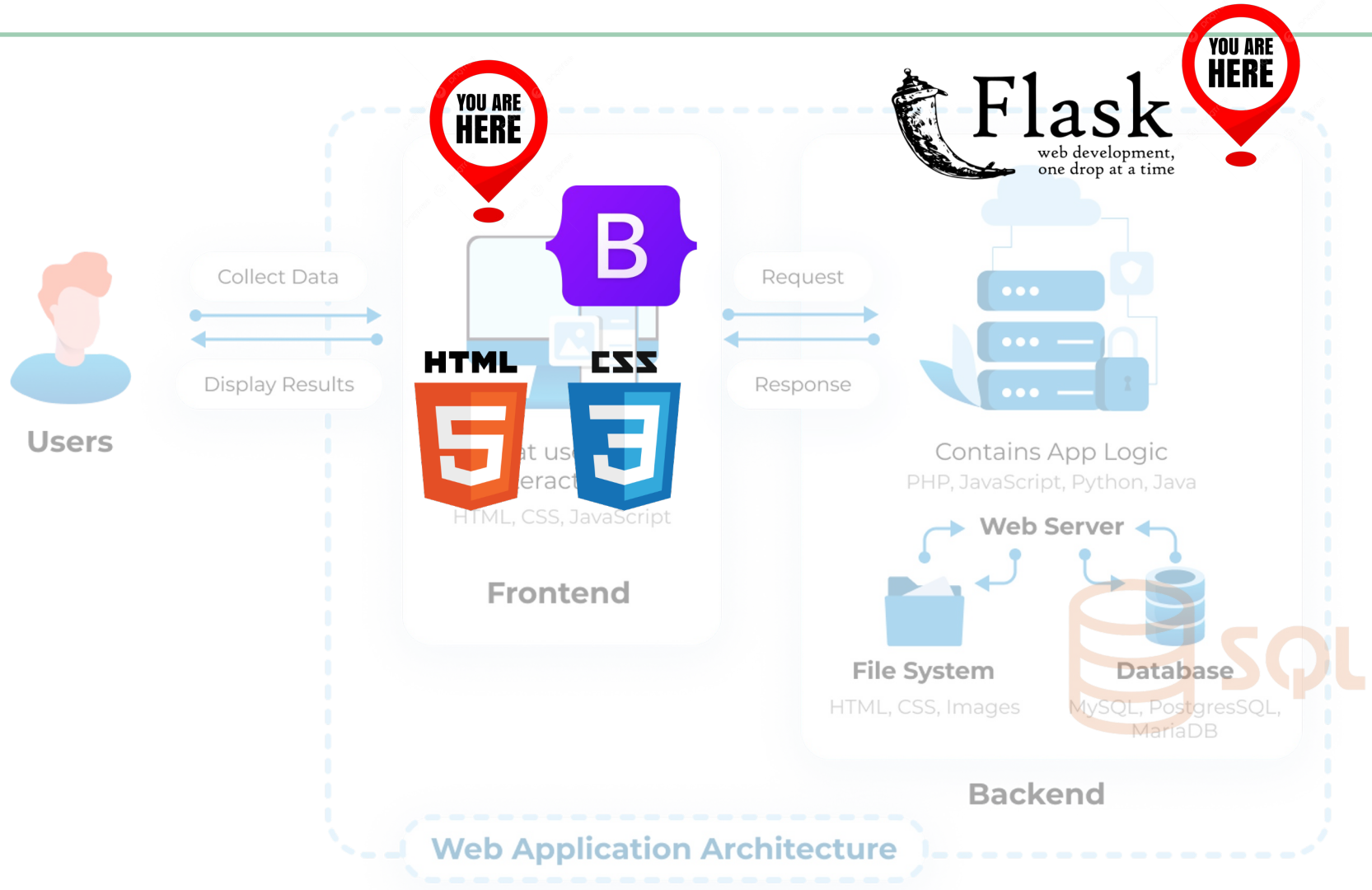
Forms

Juan Pablo Sáenz

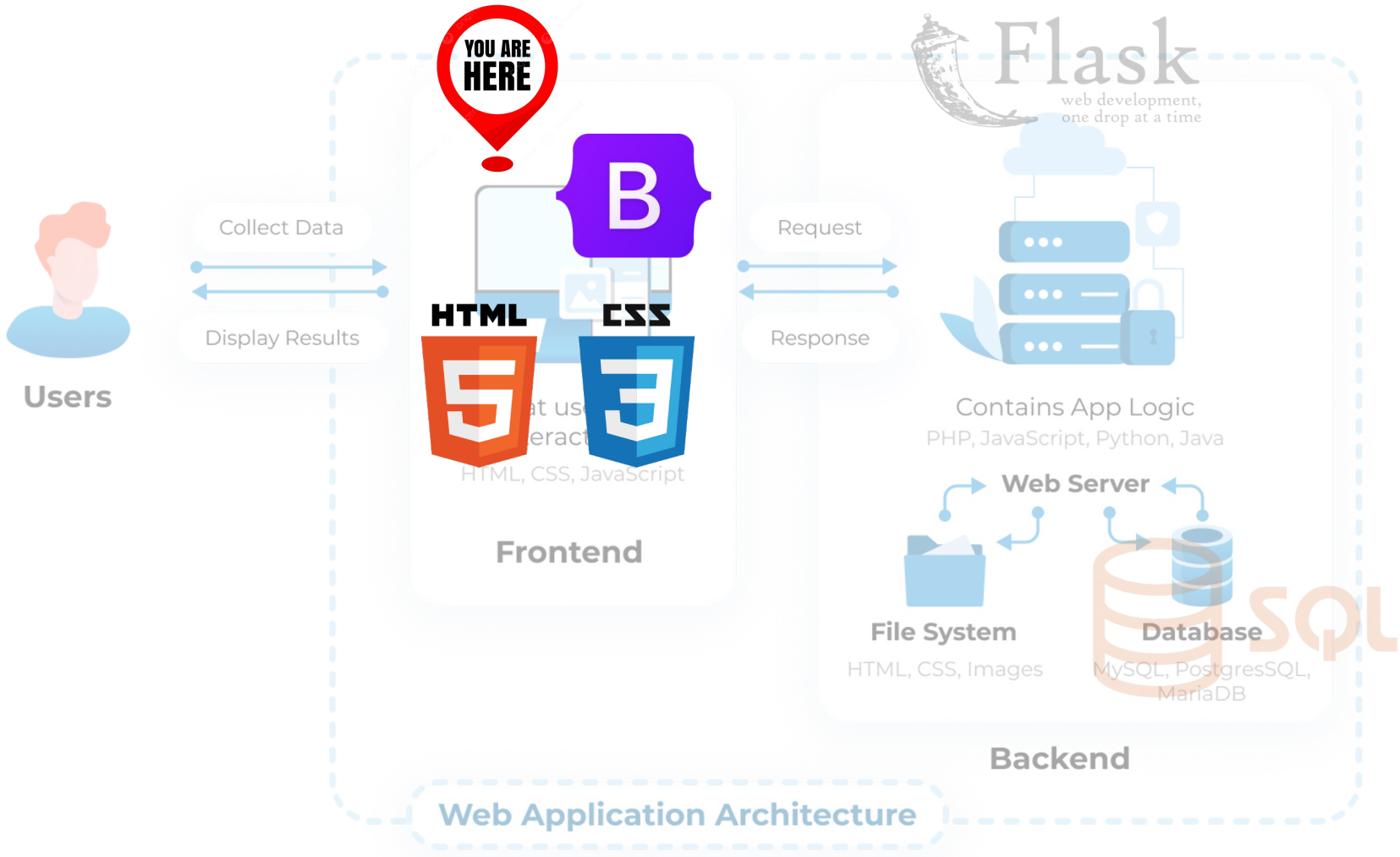
Goals

- Understand how to use **HTML5 tags for form inputs**
- Learn form **validation** techniques
- Implement form handling in **Flask**

Forms: where are we?

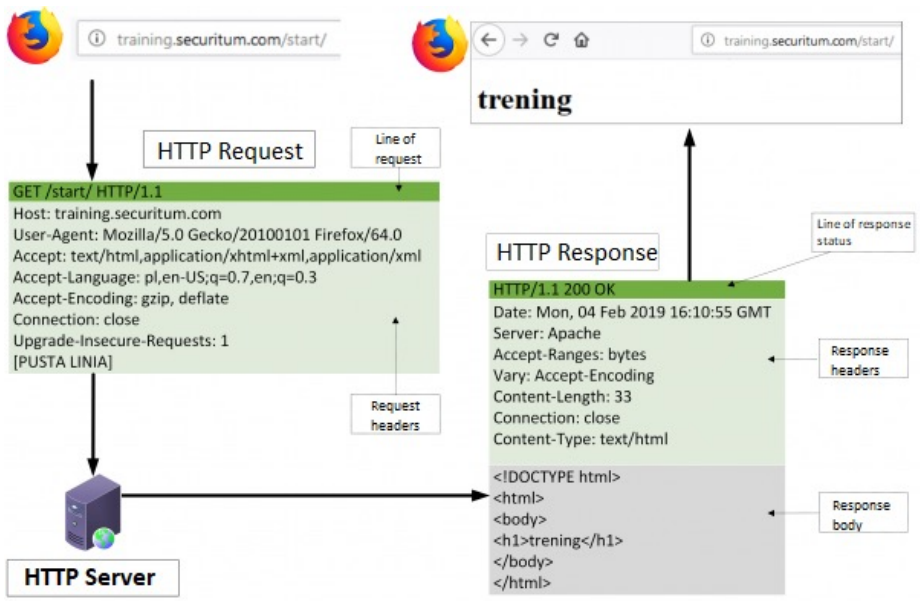


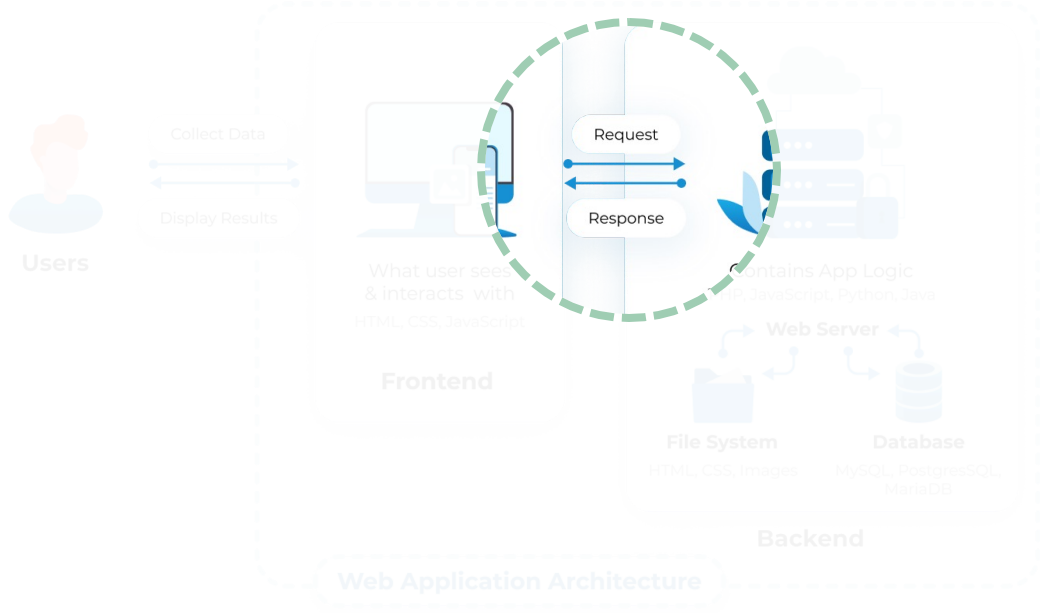
Forms in HTML



Web architecture components: HTTP Protocol

HTTP Protocol: the protocol used for **transferring data over the web**, allowing communication between **clients** (like browsers) and **servers** by sending requests and receiving responses



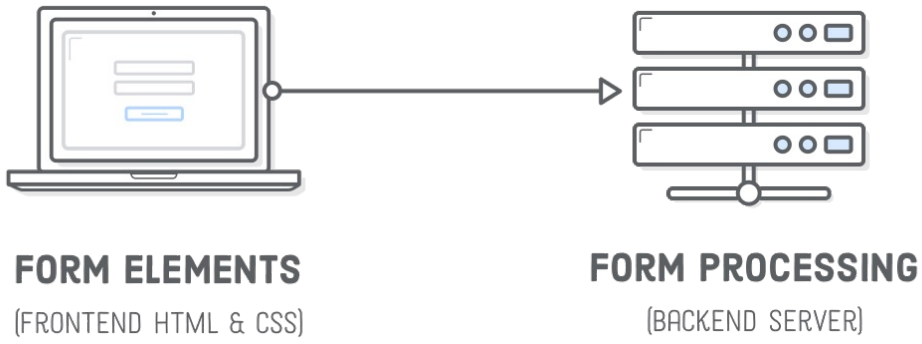


Forms

An **HTML element** that allows users to **input data**, which can be **sent to a server** for processing

- Data **collection**
- User **authentication** (login, signup)
- **Search** functionality

Forms consist of **input fields**, **labels**, **buttons**, and **validation mechanisms** to ensure correct data entry



Form declaration

`form`

- **Tag** that defines an **HTML form** for user input

`action`

- **Attribute** that specifies the **URL** where the form data should be submitted

`method`

- **Attribute** that defines the **HTTP method** for submission (default: **GET**)

```
<form action="/new-user"
method="POST" id="userdata">
<!-- Regular HTML content -->
</form>
```

HTTP methods

Define the different **types of actions** that can be performed on a resource through HTTP

- **`GET`**: Retrieve data from the server (when **fetching data** without side effects)
- **`POST`**: Send data to the server to create a new resource (when **submitting data** that should cause a change on the server)
- **`PUT`**: Update an existing resource (when **updating** or **replacing** a resource)
- **`DELETE`**: Remove a resource (when **removing** a resource)

Resource: any entity or object that can be accessed or manipulated on a server, such as data and files

HTTP methods (less commonly used)

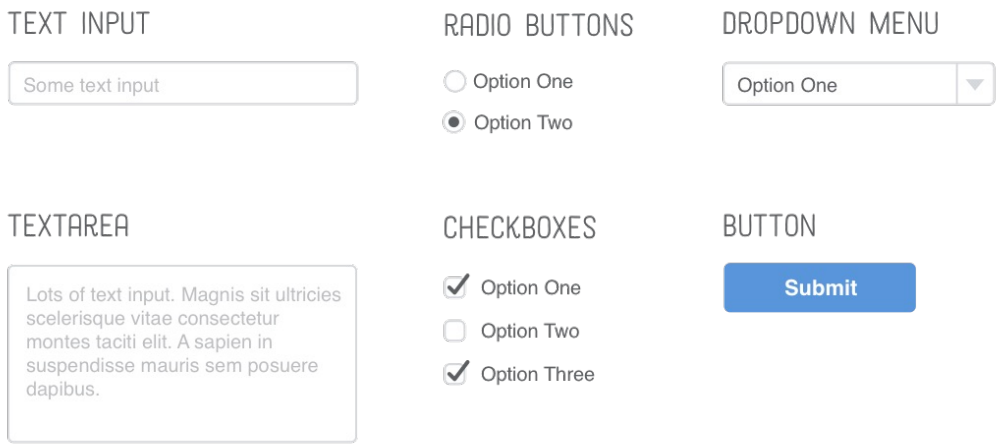
- **`PATCH`**: Apply partial modifications to a resource (when only **partial updates** are needed)
- **`OPTIONS`**: Retrieve allowed methods for a specific resource (when checking which **methods** are available for a resource)

Form controls

A form consists of various HTML elements that enable **user input** and **interaction**

Control categories:

- **Input** elements: collect user data (e.g., text, email, password)
- **Selection** elements: allow users to **choose** from **predefined options** (e.g., dropdowns, checkboxes, radio buttons)
- **Buttons**: submit or reset the form

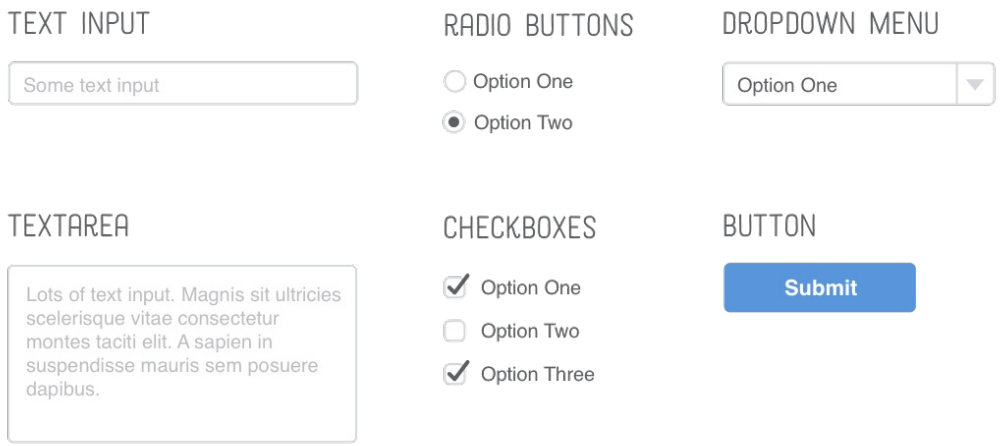


Form controls

A form consists of various HTML elements that enable **user input** and **interaction**

Support elements:

- **Labels**: describe input fields
- **Datalist**: offers predefined suggestions for input fields



Form controls: Input

Each element should have a **unique `name` attribute** for identification

The **`type` attribute** can be set to:

- `button`, `checkbox`, `password`, `email`, `date`, `color`, `number`, `month`, `file`, `hidden`, `radio`

The **`value` attribute** will hold user-provided text

```
<form action="/new-user"
method="POST" id="userdata">
<!-- Regular HTML content -->
<input type="text" name="firstname"
placeholder="Please insert your
first name"></input>
</form>
```

Common Attributes for Input Controls

checked: radio/checkbox is selected

disabled: control is disabled

readonly: value cannot be edited

size: size of the control (pixels or characters)

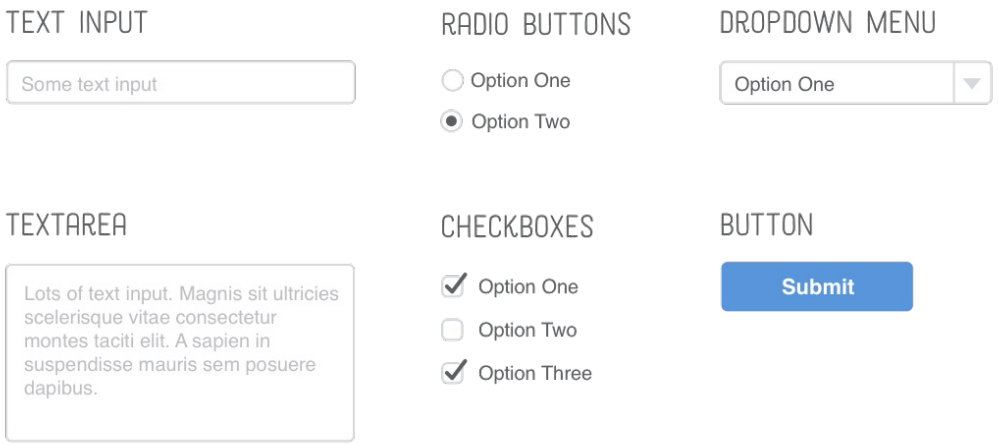
value: value entered by the user

autocomplete: hint for the browser's autofill feature

```
<input type="number" name="age"
placeholder="Your age" min="18"
max="110" />

<input type="text" name="username"
pattern="[a-zA-Z]{8}" />

<input type="file" name="docs"
accept=".jpg, .jpeg, .png" />
```

Other Form controls: Text area

<textarea>: a multi-line text field




```
<label for="story">Tell us your
story:</label>
<textarea id="story" name="story"
rows="5" cols="33"></textarea>
```

Other Form controls: Dropdown menu

<select>: Creates a **dropdown list** for selecting options

Contains multiple **<option>** elements as choices

The **value** attribute in **<option>** defines the data submitted when selected

```
<label for='t-shirt'>T-Shirt
Size</label>
<select id='t-shirt' name='t-
shirt’>
  <option value='xs'>Extra
Small</option>
  <option value='s'>Small</option>
  <option value='m'>Medium</option>
  <option value='l'>Large</option>
</select>
```

Button control

`<button>`: supports three types (**`type`** attribute) of buttons:

- **`submit`**: sends the form data to the server.
- **`reset`**: resets the form to its initial values.
- **`button`**: just a button, whose behavior needs to be specified by JavaScript

```
<button type="submit">Send</button>
<button type="reset">Clear</button>
```

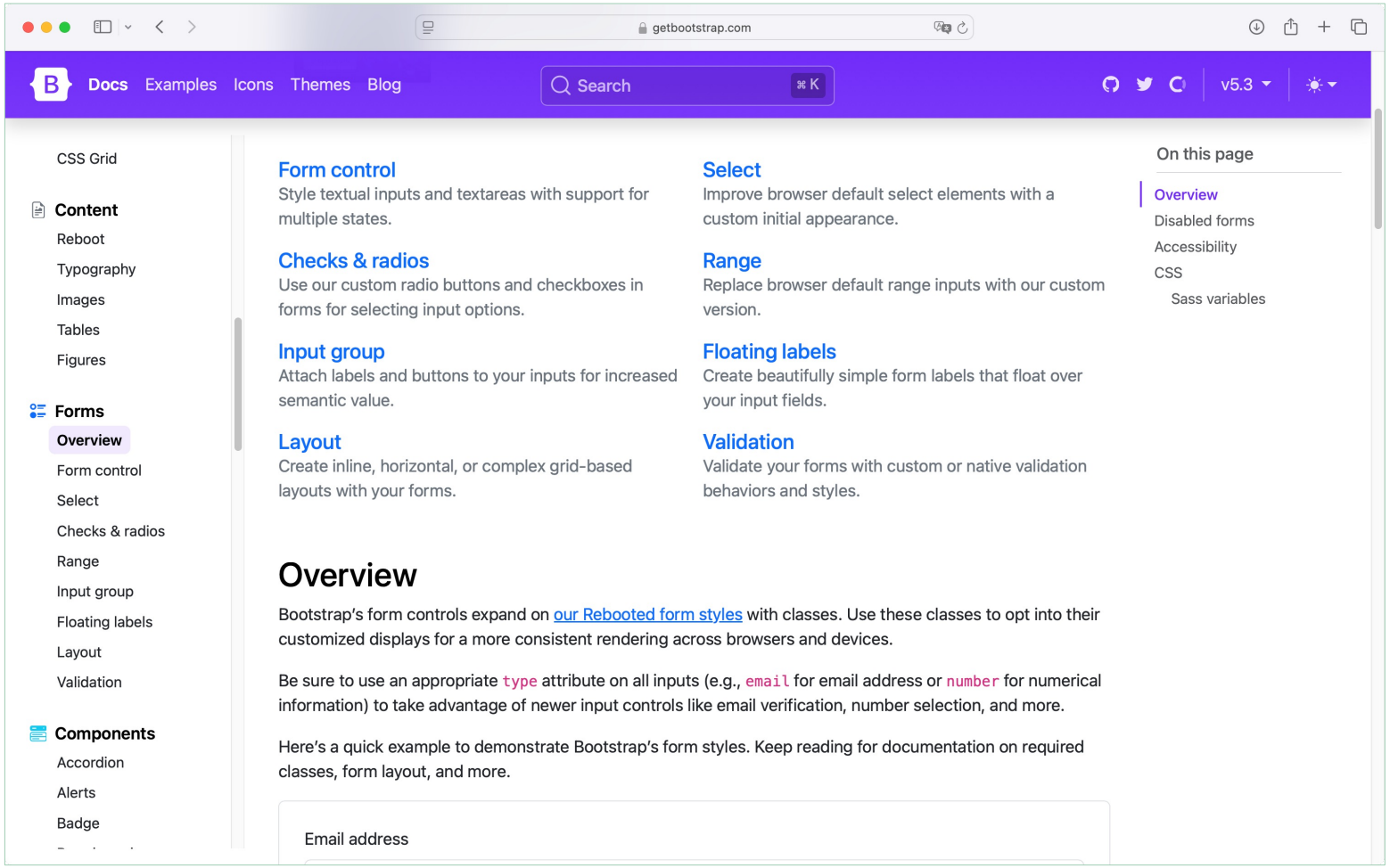
Support elements: Label tag

Represents a **caption** for a UI item

- Improves **usability**: clicking the label activates the input (useful for touch screens)
- Connects to an **`<input>`** by matching its **`id`** with the label's **`for`** attribute


```
<label for="cheese">Do you like
cheese?</label>
<input type="checkbox"
name="cheese" id="cheese">
<label for="peas">Do you like
peas?</label>
<input type="checkbox" name="peas"
id="peas">
```

Forms in Bootstrap



Let's see it in practice

Form validation

Ensures data is in the correct **format** and meets **application constraints**

Two types of validation:

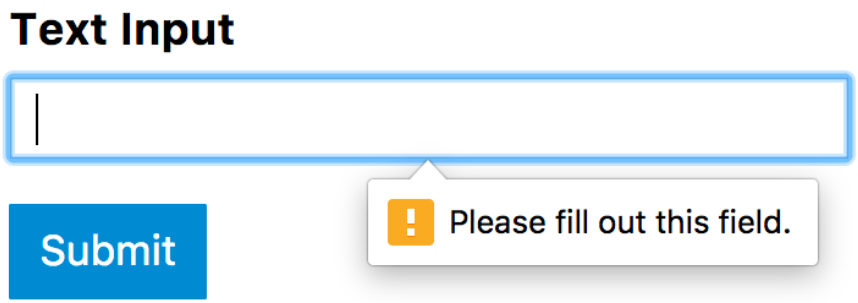
- **Client-side**: performed in the **browser** using HTML5 and JavaScript
- **Server-side**: handled by the **application server**

Client-side validation

Checks input **before submission**, and after passing, data is sent to the server for processing

- **Protects** user data (e.g., enforcing secure passwords)
- Improves user experience with **immediate feedback**

⚠ **NEVER** trust client-side validation on the server!



Client-side validation

Built-in HTML5 Form Validation

- **`type="email"`**: ensures the value follows email syntax
- **`type="url"`**: ensures the value is a properly formatted URL
- **`type="number"`**: restricts input to numeric values
- **`required`**: prevents form submission if the field is empty
- **`pattern="[A-Za-z]{3,}"`**: enforces a custom regex pattern (e.g., at least three letters)
- **`minlength`**/**`maxlength`**: sets the minimum and maximum length for text input
- **`min`**/**`max`**: defines the allowable numerical range for number inputs

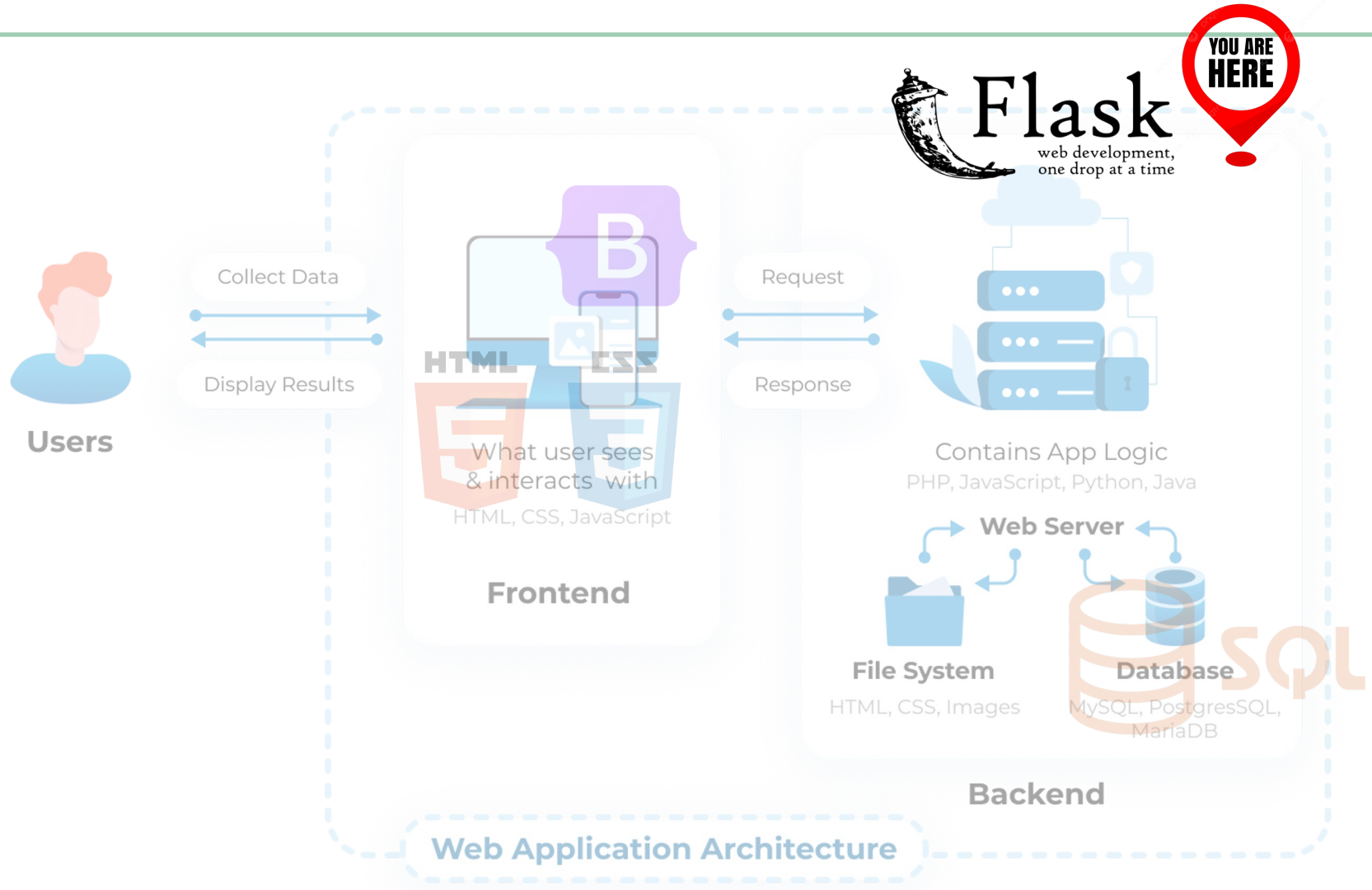
Client-side validation

```
<form>
  <label for="email">Email:</label>
  <input type="email" id="email" name="email" required>

  <label for="website">Website:</label>
  <input type="url" id="website" name="website" required>

  <label for="username">Username (3-10 letters):</label>
  <input type="text" id="username" name="username" pattern="[A-Za-z]{3,10}" minlength="3"
maxlength="10" required>

  <label for="age">Age (18-99):</label>
  <input type="number" id="age" name="age" min="18" max="99" required>

  <button type="submit">Submit</button>
</form>
```

Client-side validation

```
<form>
  <label for="email">Email:</label>
  <input type="email" id="email" name="email" required>

  <label for="website">Website:</label>
  <input type="url" id="website" name="website" required>

  <label for="username">Username (3-10 letters):</label>
  <input type="text" id="username" name="username" pattern="[A-Za-z]{3,10}" minlength="3"
maxlength="10" required>

  <label for="age">Age (18-99):</label>
  <input type="number" id="age" name="age" min="18" max="99" required>

  <button type="submit">Submit</button>
</form>
```

Forms in Flask



Handling form data in Flask

Form submission

- The entire form content is sent via an HTTP request (**POST** or **PUT**) to the application server

Accessing Form Data in Flask

- Flask stores form data in **`request.form`**
- **`request.form`** behaves like a **dictionary**, where **keys** are input field **names**

```
from flask import Flask,
render_template, request

# Access a specific field
name = request.form['name']
# Safer method (avoids KeyError)
email = request.form.get('email')
```

Handling form data in Flask

This route handles POST requests at **/subscribe**

@app.route("/subscribe", methods=["POST"])

- Defines the route for handling POST requests

```python
# app.py
@app.route("/subscribe", methods=["POST"])
def add_to_mailing_list():
  name = request.form.get("name")
  email = request.form.get("email")
  return f"Added to mailing list: {name}, {email}"
```

```html
<!-- In the HTML file -->
<form action="/subscribe" method="post">
  <input type="text" id="name" name="name" required>
  <input type="email" id="email" name="email" required>
  <button type="submit">Subscribe</button>
</form>
```

Handling form data in Flask

`request.form.to_dict()`

- Converts the submitted form data into a standard Python dictionary (**optional**)

```
# Suppose the form sends
name="Alice" and
email=alice@example.com
recensione = request.form.to_dict()
print(recensione)
# Output: {'name': 'Alice',
'email': 'alice@example.com'}
```

Server-side validation

Checks for **empty names** or **emails**

Check if the **email** contains "**@**"

Flask provides pre-configured **logging facilities**, ready to use:

- **`app.logger.debug`**
- **`app.logger.warning`**
- **`app.logger.error`**

```
# Validation
if not name or not email:
  app.logger.warning("Form submitted
with missing fields.")
elif "@" not in email:
  app.logger.warning("Form submitted
with invalid email: %s", email)
else:
  app.logger.info("User subscribed
successfully: %s", email)
```

Redirect in Flask

Flask provides the **`redirect()`** function to redirect users to a **different URL**

Commonly used after **form submissions** or when **handling user authentication**

- ✅ **`redirect()`** changes the URL
- **`render_template()`** does not affect the URL; it just renders the content associated with a route

```
from flask import Flask, render_template,
request, redirect

# Validation
if not name or not email:
  app.logger.warning("Form submitted with missing
fields.")
  return redirect(url_for('show_error'))
elif "@" not in email:
  app.logger.warning("Form submitted with invalid
email: %s", email)
  return redirect(url_for('show_error'))
else:
  app.logger.info("User subscribed successfully:
%s", email)
  return redirect(url_for('home'))
```

File uploads

- Forms for uploading files must include the **`enctype="multipart/form-data"`** attribute
- The original filename (with extension) is available in the **filename** attribute

```
# app.py
uploaded_file = request.files['file']
# Save file
uploaded_file.save('uploads/' + uploaded_file.filename)
```

```
<!-- In the HTML file -->
<form action="/upload" method="POST"
enctype="multipart/form-data">
  <label for="file">Choose file to upload:</label>
  <input type="file" id="file" name="file">
  <button type="submit">Upload</button>
</form>
```

File Uploads: Renaming Files

Avoid filename collisions and enhance security

- Multiple users may upload files with the same name
- Uploaded filenames may contain malicious patterns or special characters
- **`secure_filename()`** to sanitize user-uploaded filenames before saving them on the server

```python
from werkzeug.utils import secure_filename
import time

original_filename =
secure_filename(uploaded_file.filename)
new_filename =
f"{int(time.time())}_{original_filename}"
uploaded_file.save(f"uploads/{new_filename}")
```

Licenza

- These slides are distributed under a Creative Commons license “**Attribution-NonCommercial-ShareAlike 4.0 International (CC BY-NC-SA 4.0)**”
- **You are free to:**
  - **Share** — copy and redistribute the material in any medium or format
  - **Adapt** — remix, transform, and build upon the material
  - The licensor cannot revoke these freedoms as long as you follow the license terms.
- **Under the following terms:**
  - **Attribution** — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
  - **NonCommercial** — You may not use the material for commercial purposes.
  - **ShareAlike** — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.
  - **No additional restrictions** — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.
- https://creativecommons.org/licenses/by-nc-sa/4.0/