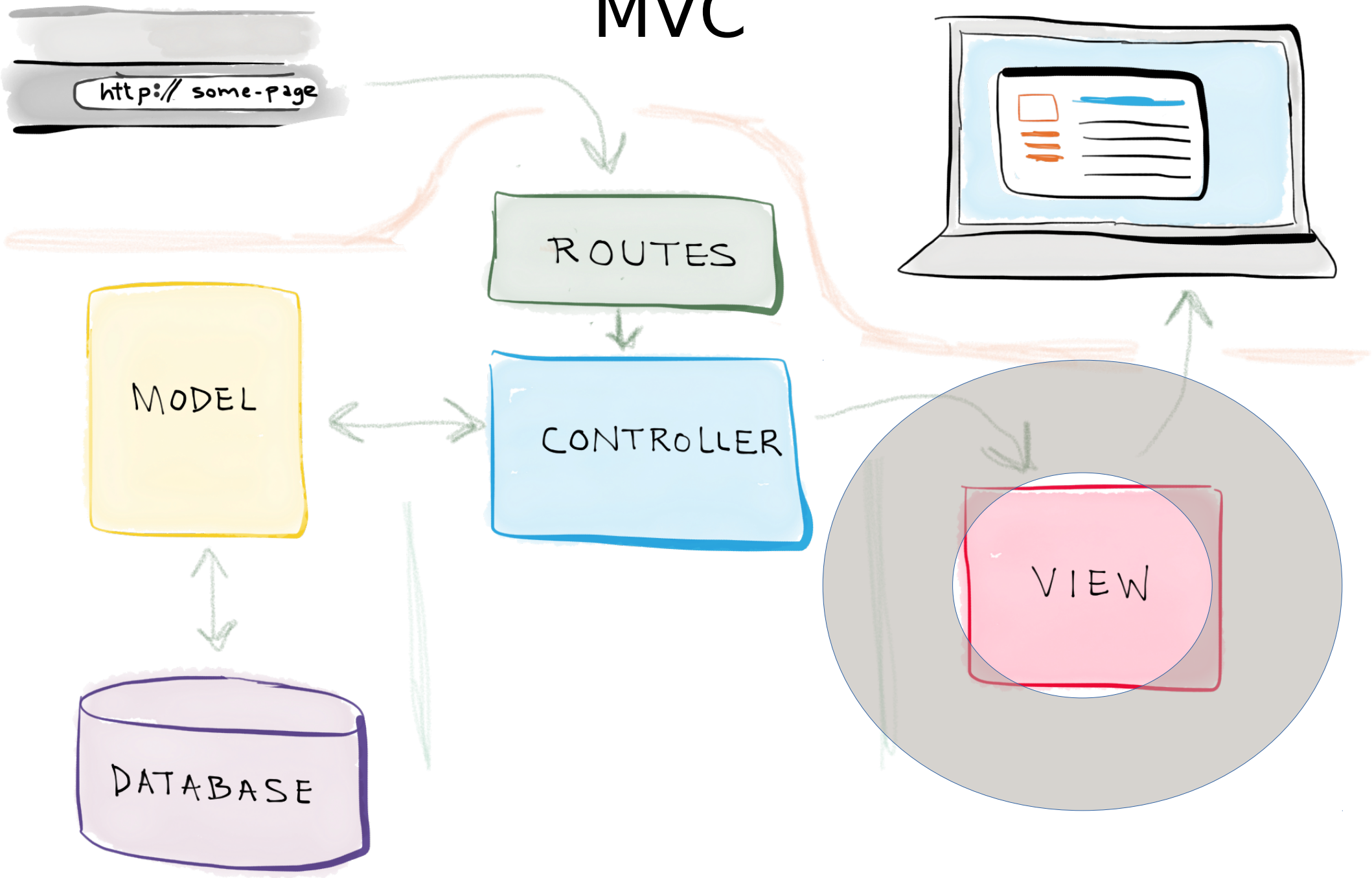
MVC



Templates (why?)

Problem with generating HTML directly with code

- Easy to make very hard to read code
- Mix-up the control code with the view code

Text template system is a convenient and common way to separate the view code from the remaining code

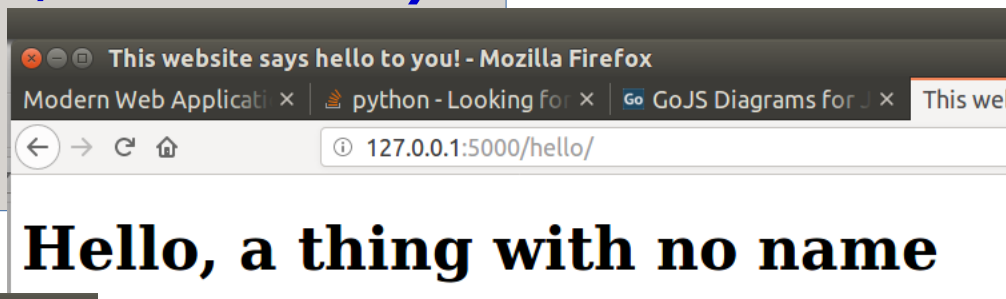
- Flask uses Jinja2 as template system.

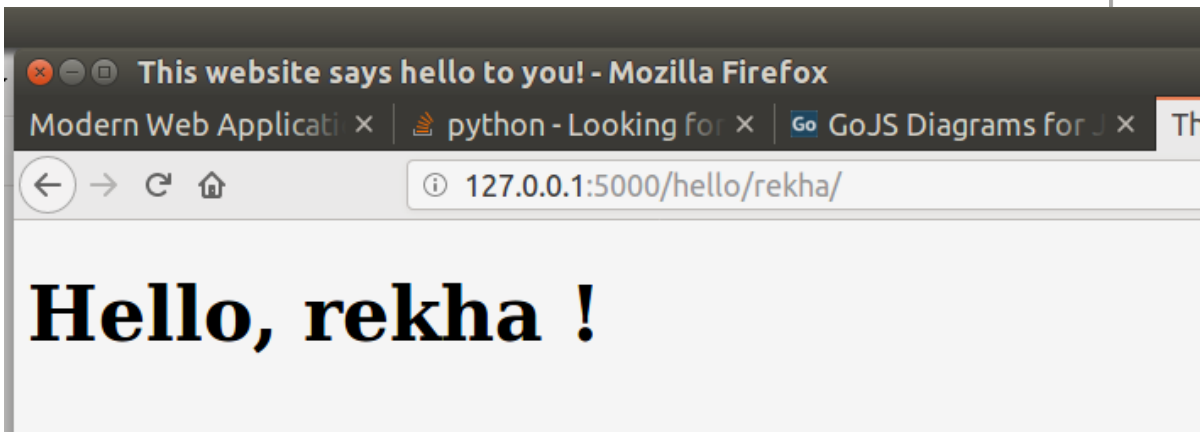
Basic Template Rendering

render_template

```
from flask import Flask, render_template

#create the application.
app = Flask(__name__)

@app.route('/hello/')
@app.route('/hello/<user>/')
def hello(user = None):
    return render_template('hello.html', name=user)

if __name__ == '__main__':
    app.run(debug=True)
```

Takes a path to an HTML file,and arbitrary parameters and returns contents of html





Basic Template Rendering

{{}}, {%---%}

```
<!doctype  html>
 <head>
  <title> This website says hello to you! </title>
 </head>
 <body>
    {% if name %}
    <h1>Hello, {{name}} !</h1>
    {% else %}
    <h1>Hello, a thing with no name</h1>
    {% endif %}
  </body>
  </html>
```

It's no ordinary HTML ⇒ there are instruction mixed in !

The name variable here passed as a parameter of render_template

- hello.html is processed to generate the HTML to send to a user.
- Variables values can be rendered to text with {{ }}
- Blocks of code are put between{% %}

Templates

Flask assumes that all your templates will be in a template directory, relative to your script

```
|– templates
        |–h e l l o . html
|– t e s t . py
```

Templates using resources

If you wish to use other file ressources, like pictures or CSS files, you can put them in directory named static

```
|– templates
          |–hello.html
|– static
          |– style.css
|– test.py
```

These resources are not dynamic, not generated on the fly like the HTML code, hence the name ”static”.

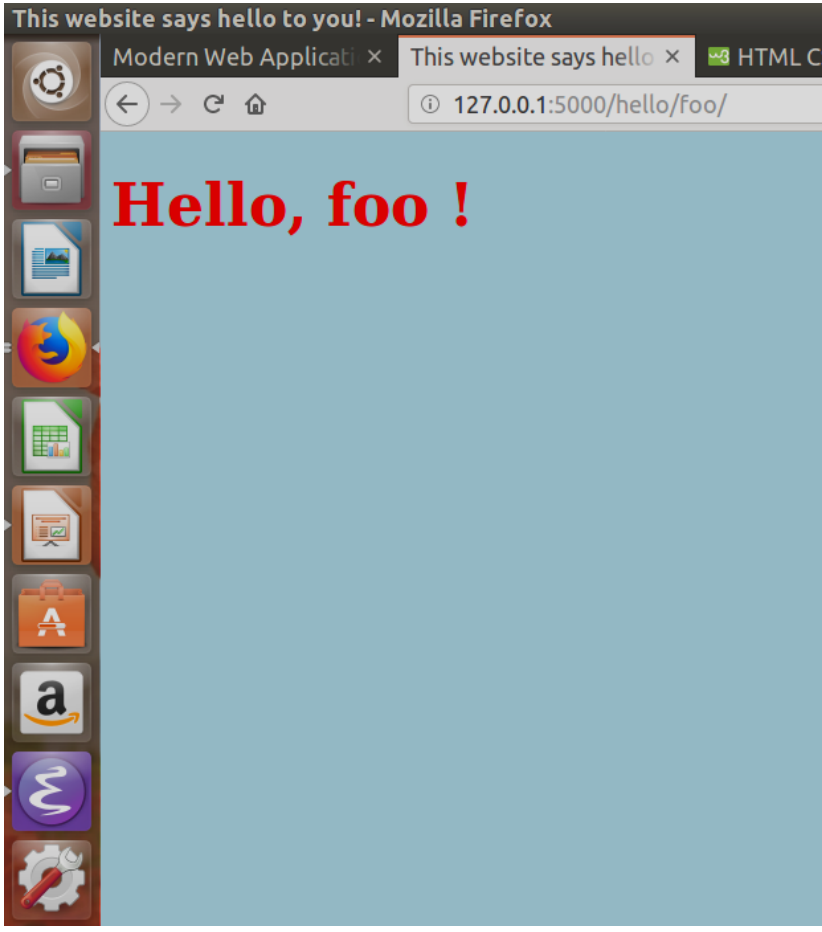
Templates using resources

Then, to use those resources, you can again use url_for

<link rel=stylesheet type=text/css href="{{url_for('static', filename='style.css')}}"

Contents of static/style.css:

```
body
{
    background-color:lightblue;
}

h1
{
    color:red;
}
```



Templates

inheritence

Jinja2 provides a simple way to share a common template and specialize it :template inheritance

```
{% extends 'base.html' %}
 {% block content %}
    {% if name %}
    <h1>Hello, {{name}} !</h1>
    {% else %}
    <h1>Hello, a thing with no name</h1>
    {% endif %}
 {% endblock %}
```

Above template extends base.html

Templates inheritence

```
<!doctype  html>
 <head>
   <title> This website says hello to you! </title>
    <link rel=stylesheet type=text/css
href="{{url_for('static', filename='style.css')}}"
 </head>
<body>
  <div id = "container">
    <div id = "header">
      <h2>salute.com</h2>
      <p> This website salutes you!</p>
    </div>
    <div id  = "content">
      {% block content%}
      {% endblock %}
    </div>
  </div>
  <div id = "footer">
    <h2> salute.com </h2>
    <p> copyright &copy; Foo Bar </p>
  </div>
</body>
</html>
```

- This template defines code which is common across all the templates. Children templates can inherit from it.
- Serves as a parent template
- A base template defines the empty block “content” which needs to be provided by children template.

```
{% block content%}
{% endblock %}
```

- Children template define the same block and add code to it.

```
{% block content%}
<h1>some stufff </h1>
{% endblock %}
```

Templates

inheritence- look and feel across multiple views

```
{% extends 'base.html' %}
 {% block content %}
    {% if name %}
    <h1>Hello, {{name}} !</h1>
    {% else %}
    <h1>Hello, a thing with no name</h1>
    {% endif %}
 {% endblock %}
```

Templates

inheritence- look and feel across multiple views

```
{% extends 'base.html' %}
 {% block content %}
    {% if name %}
    <h1 style="color:blue;">Welcome, {{name}} !</h1>
    {% else %}
    <h1 style="color:=blue;">Welcome, a thing with no name</h1>
    {% endif %}
 {% endblock %}
```

Templates

inheritence

In this exemple, extending base.html provides

Coherent look, code reusage, and clean separation

- A common title
- Includes common ressources (css, javascript, etc.)
- A common header
- A common footer
- The specialized part goes in the ”content” block.

Templates macros

We can define reusable HTML bits of code using macros.

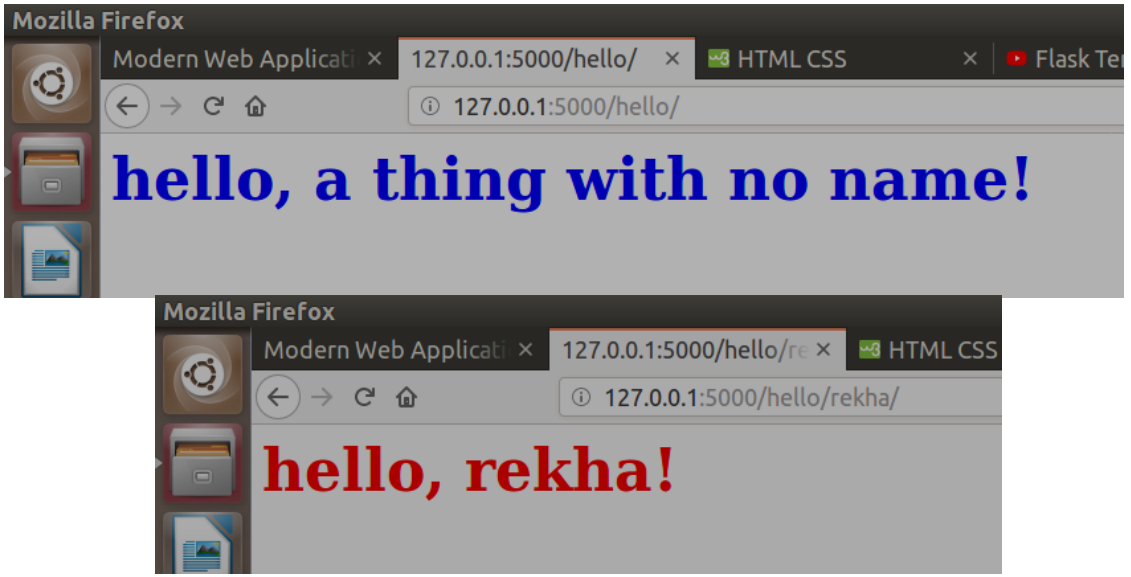
Macros are functions for templates.

```
{% macro greet(greeting_word, name) %}
  {% if name %}
   <h1 style="color:red;">{{greeting_word}}, {{name}}!</h1>
  {% else %}
   <h1 style="color:blue;">{{greeting_word}}, a thing with no name!
</h1>
  {% endif %}
{% endmacro %}
```

Templates macros

Use macros in templates:

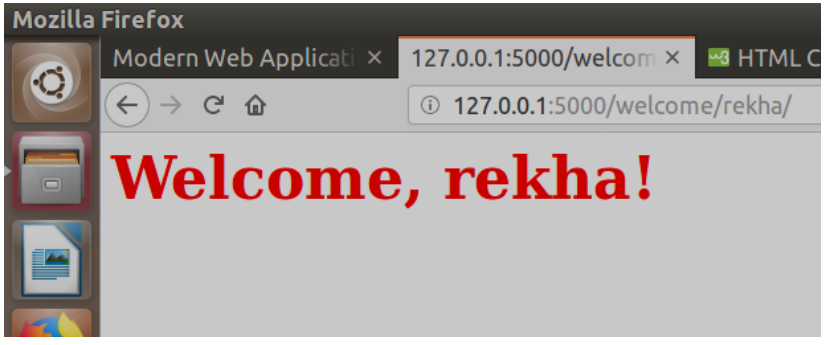
```
{% import "macro.html" as col %}
<html>
  <body>
    {{col.greet('hello', name)}}
  </body>
</html>
```



Templates macros

Use macros in templates:

```
{% import "macro.html" as col %}
<html>
  <body>
    {{col.greet('Welcome', name)}}
  </body>
</html>
```



Templates language

Jinja templates use their own language, more or less Python-like.

- It tries to imitate Python but it is not Python
- Jinja provides a limited language because

  It’s a view.

  No business code here.

  Just HTML generation.
- Should be fast as It’s a page that might be served for many different users.

Templates language

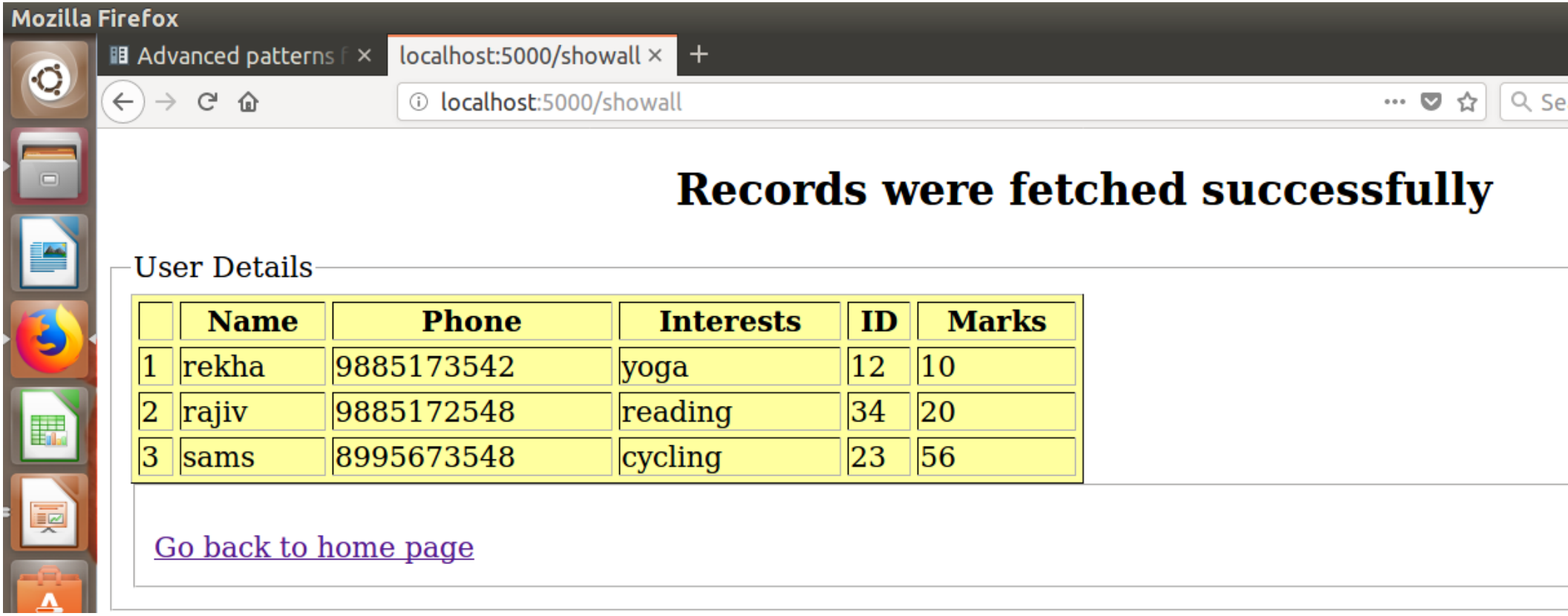
- If-elif-else-endif
- for-endfor
- for-if-endfor(filter)
- loop object
- for-else-endfor

Templates language

```
<!doctype html>
<html>
  <h2 style = "text-align: center;">{{ message }}</h2>
  <body link="blue">
   <fieldset>
     <legend>User Details</legend>
     <table border = 1 cellspacing=5 width=50% height=50px bgcolor="#ffff9f">
       <tr>
           <th></td>
          <th>Name</td>
          <th>Phone</td>
          <th>Interests</td>
          <th>ID</td>
            <th>Marks</td>
       </tr>
      {% for row in rows if row["id"] %}
            <tr>
              <td>{{loop.index}}</td>
              <td>{{row["name"]}}</td>
              <td>{{row["phone"]}}</td>
              <td>{{row["interests"]}}</td>
              <td>{{row["id"]}}</td>
             <td>{{row["marks"]}}</td>
            </tr>
        {% else %}
          <h1>no user found!</h1>
        {% endfor %}
     </table>
     <fieldset>
     <br><a href = "/">Go back to home page</a>
  </body>
</html>
```

Templates language

```
@app.route('/showall')
def showall():
   rows,msg = model.getAll()
   return render_template('showall.html', rows=rows, message=msg)
```



Requests(object)

- To access incoming request data, you can use the global request object.
- Flask parses incoming request data for you and gives you access to its proxy through that global object.
- Internally Flask makes sure that you always get the correct data for the active thread if you are in a multithreaded environment.

```
@app.route('/addrec',methods = [‘GET’, 'POST'])
def addrec():
   if request.method == 'POST':
      res, msg = model.addUser(request)
      return render_template("result.html", result=res, message=msg)
   else:
      return render_template("adduser.html")
```

Requests(get/post)

We can send data (HTML, JSON, XML, any kind of text), but we also need to receive data.

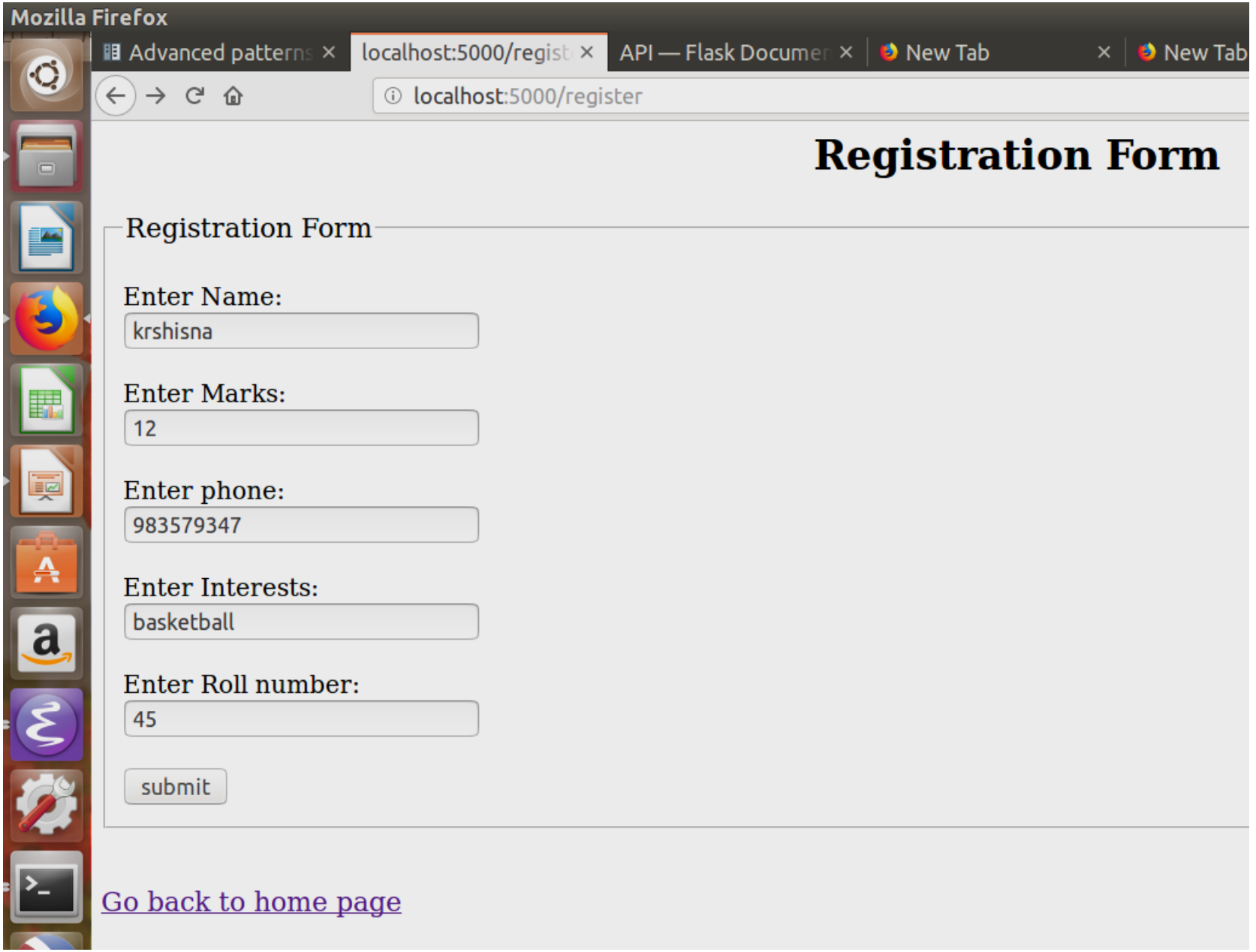
The HTTP protocol defines different kind of requests:

- GET⇒request to server to send data after receiving data in url arguments
- POST⇒request to server to send data after accepting sent data

Requests adduser.html with post

```
<html>
  <h2 style = "text-align: center;">Registration Form</h2>
   <body>
     <form action = "http://localhost:5000/addrec" method = "post">
       <fieldset>
         <legend>Registration Form</legend>
         <div style = font-size:20px; font-weight:bold; margin-left:150px;><br>
          Enter Name:<br> <input type = "text" name = "nm" required/>
         </div>
         <div style = font-size:20px; font-weight:bold; margin-left:150px;><br>
           Enter Marks:<br> <input type = "text" name = "mrks" />
         </div>
         <div style = font-size:20px; font-weight:bold; margin-left:150px;><br>
             Enter phone:<br> <input type = "text" name = "phone" required/>
         </div>
         <div style = font-size:20px; font-weight:bold; margin-left:150px;><br>
            Enter Interests:<br> <input type = "text" name = "interests"/>
         </div>
         <div style = font-size:20px; font-weight:bold; margin-left:150px;><br>
          Enter Roll number:<br> <input type = "text" name = "id" required/>
         </div>
         <div style = font-size:20px; font-weight:bold; margin-left:150px;><br>
            <input type = "submit" value = "submit" />
         </div>
         </fieldset>
       </form>
       <br><a href = "/">Go back to home page</a>
    </body>
</html>
```

Requests adduser.html with post

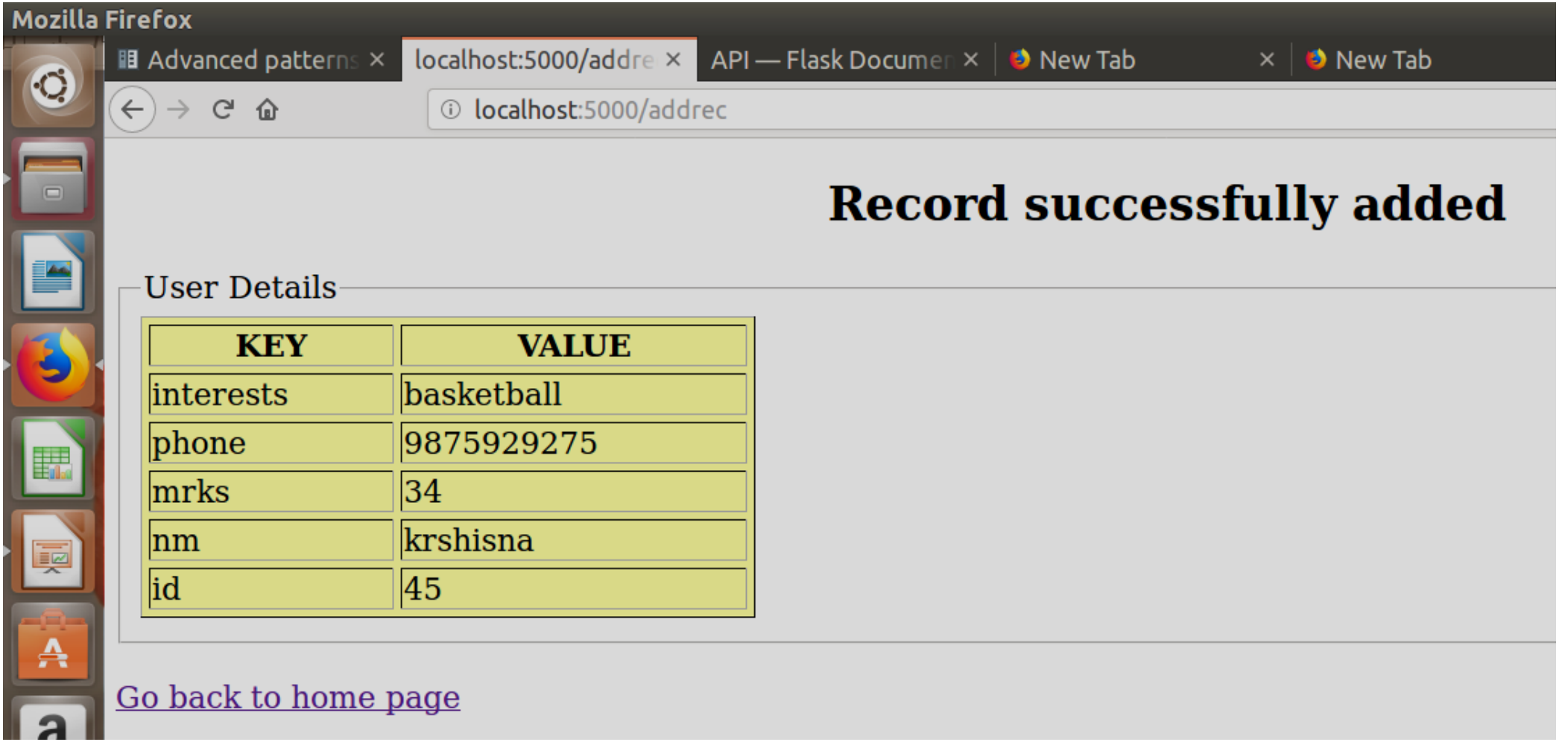


Requests adduser.html with post

```
@app.route('/addrec',methods = ['POST', 'GET'])
def addrec():
   if request.method == 'POST':
      res, msg = model.addUser(request)
      return render_template("result.html", result=res, message=msg)
   else:
      return render_template("adduser.html")
```

After submitting the above form =>

Requests adduser.html with post



Requests adduser.html with get

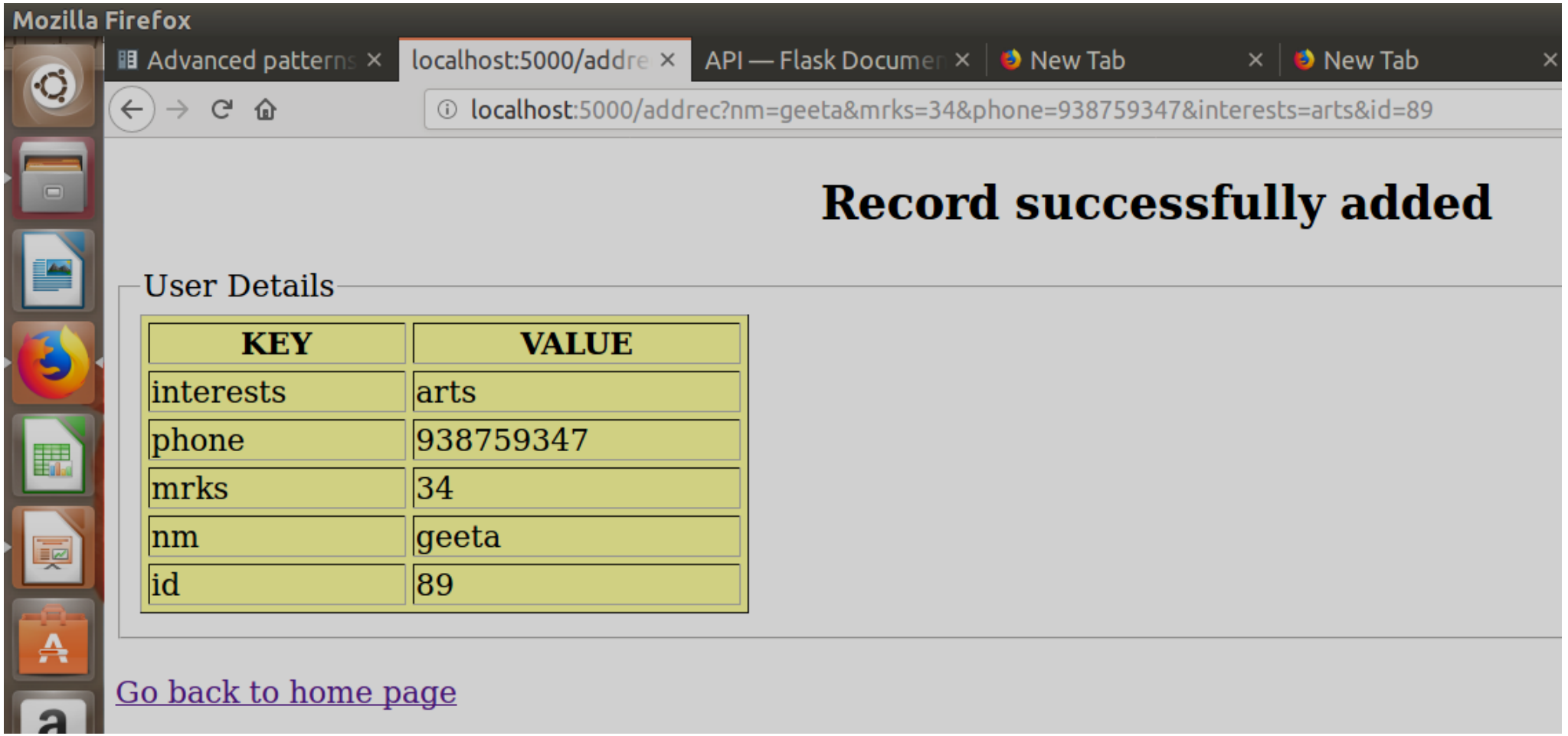
```
<html>
  <h2 style = "text-align: center;">Registration Form</h2>
   <body>
     <form action = "http://localhost:5000/addrec" method = "get">
```

```
@app.route('/addrec', methods = ['POST', 'GET'])
def addrec():
   res, msg = model.addUser(request)
   return render_template("result.html", result=res, message=msg)
```

After submitting the above form =>

Requests adduser.html with get

http://localhost:5000/addrec?
nm=geeta&mrks=34&phone=938759347&interests=arts&id=89



Requests(get/post)

- The request object hold the information sent to the server
- request object provides all of the attributes Werkzeug defines plus a few Flask specific ones. 2 useful ones are below

form

A MultiDict with the parsed form data from POST or PUT requests. Please keep in mind that file uploads will not end up here, but instead in the files attribute.

user = request.form['nm']

args

A MultiDict with the parsed contents of the query string. (The part in the URL after the question mark, get parsed in GET request).

user = request.args['nm']

MVC