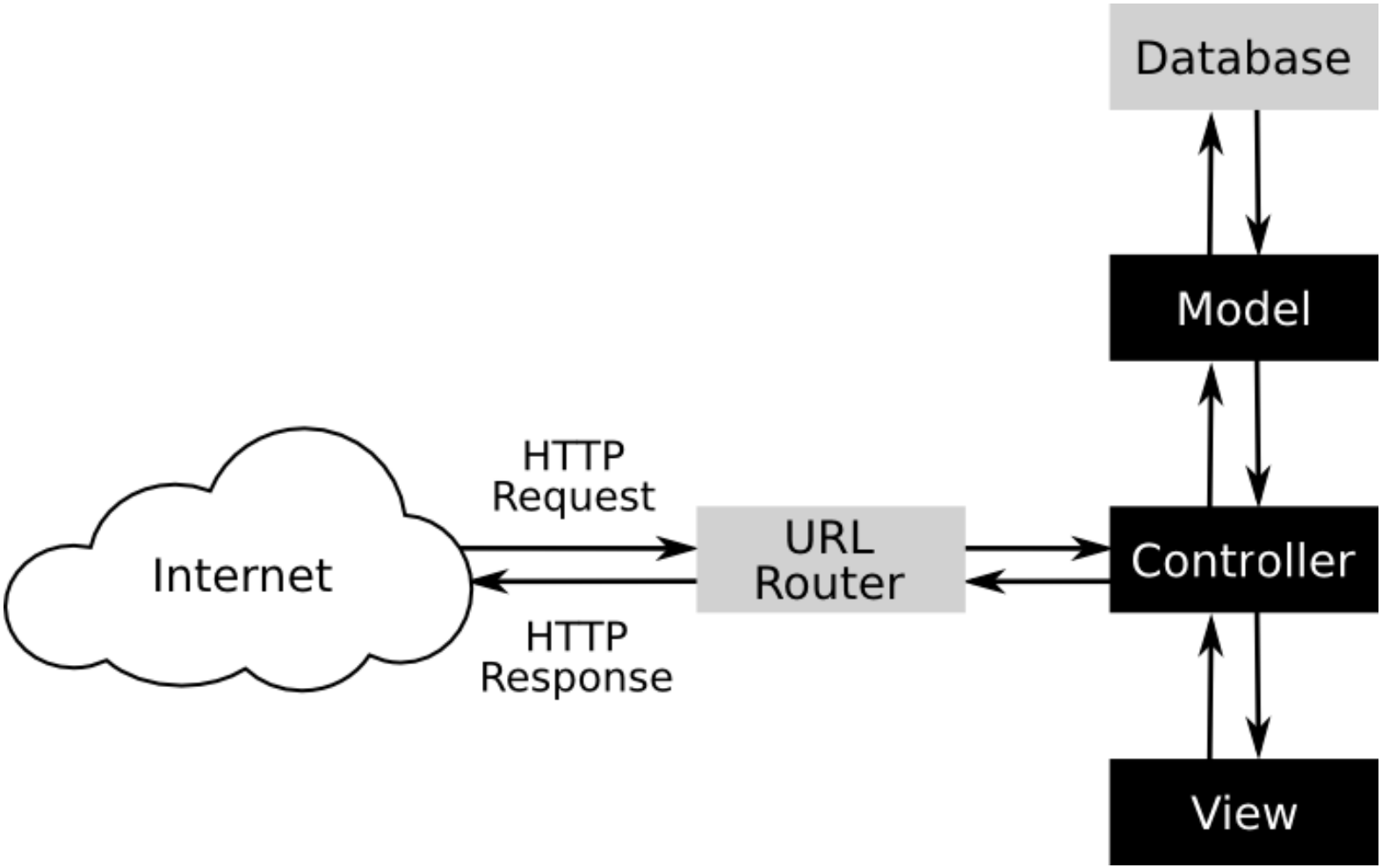
Web Frameworks and MVC

Daniel Zappala

CS 360 Internet Programming
Brigham Young University

Model View Controller



Controller

```python
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello():
    return 'Hello World!'

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

- from the Flask microframework
- maps the route "/" to the method hello()

Model

```
1  @app.route('/')
2  def show_entries():
3      db = get_db()
4      cur = db.execute('select title, text from entries order by id desc')
5      entries = cur.fetchall()
6      return render_template('show_entries.html', entries=entries)
```

- from the flaskr example blog app
- fetches blog entries from the database
  - integrates model into controller
  - delivers entries to the view for rendering
- many frameworks include an ORM to provide an object-oriented wrapper around the database

View

```
1  {% extends "layout.html" %}
2  {% block body %}
3    {% if session.logged_in %}
4      <form action="{{ url_for('add_entry') }}" method=post class=add-
           entry>
5        <dl>
6          <dt>Title:
7          <dd><input type=text size=30 name=title>
8          <dt>Text:
9          <dd><textarea name=text rows=5 cols=40></textarea>
10         <dd><input type=submit value=Share>
11       </dl>
12     </form>
13   {% endif %}
14   <ul class=entries>
15   {% for entry in entries %}
16     <li><h2>{{ entry.title }}</h2>{{ entry.text|safe }}
17   {% else %}
18     <li><em>Unbelievable. No entries here so far</em>
19   {% endfor %}
20   </ul>
21 {% endblock %}
```

View

- Flask uses Jinja2 for templates
  - HTML with a subset of Python for variables, control
- inheritance
  - extends the layout.html template
  - code in `block body` placed in corresponding block in layout.html
- styling provided by CSS

View

```
1  <!doctype html>
2  <title>Flaskr</title>
3  <link rel=stylesheet type=text/css href="{{ url_for('static', filename='style.
       css') }}">
4  <div class=page>
5    <h1>Flaskr</h1>
6    <div class=metanav>
7    {% if not session.logged_in %}
8      <a href="{{ url_for('login') }}">log in</a>
9    {% else %}
10     <a href="{{ url_for('logout') }}">log out</a>
11   {% endif %}
12   </div>
13   {% for message in get_flashed_messages() %}
14     <div class=flash>{{ message }}</div>
15   {% endfor %}
16   {% block body %}{% endblock %}
17 </div>
```

Example Code

▸ Citizen Budget

- Flask
- SQLAlchemy, an ORM for relational databases
- JQuery
- Highcharts for graphs

Web Frameworks

I cannot possibly cover all web development frameworks.

This is a biased sample of the best available choices.

Python

Flask

▸ Flask

- microframework
  - single file development, extensible to multiple files
  - built in development server and debugger
  - integrated unit testing support
  - RESTful request dispatching
  - uses Jinja2 templating
- ▸ SQLAlchemy provides ORM for relational DB
- various libraries for interacting with document-oriented (JSON) DBs
  - ▸ MongoAlchemy
  - ▸ MongoKit
  - ▸ PyMongo
- additional libraries for Markdown support, etc.

Django

▸ Django

- complete framework
  - separate declaration of routing instead of marking up controllers
  - separates models, views, controllers into separate directories
  - templates similar to Jinja2
  - includes an ORM
  - automatic admin interface
  - internatonalization
- see the ▸ tutorial for examples

Ruby

Sinatra

Sinatra

- microframework
  - domain specific language
  - supports many diffrent templates
  - uses Rack for middleware, eg. logging, debugging, routing, authentication, session handling, testing

```
1  require 'sinatra'
2
3  get '/' do
4    "Hello World!"
5  end
```

Padrino

▸ Padrino

- micro+ framework
  - built on top of Sinatra
  - support for testing, templating, mocking, DB libraries
  - various helpers
  - mailer
  - caching
  - admin interface with authentication
  - unified logger
  - localization
- ▸ Screencast

Rails

- Rails
  - complete framework
    - convention over configuration, DRY (Don't Repeat Yourself)
    - controllers and routing
    - includes an ORM, with migrations (Active Record)
    - built in views (Action View) that support HTML sprinkled with Ruby
    - mailer, internatonalization, testing, security, debugging, caching,asset pipeline, Javascript, plugins
  - Screencasts
  - tutorial

Javascript

node.js

node.js

- server-side code
- event-driven, non-blocking I/O

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/plain'});
  res.end('Hello World\n');
}).listen(1337, '127.0.0.1');
console.log('Server running at http://127.0.0.1:1337/');
```

express

▸ express

- web application framework for node
- use ▸ mongoose for a MongoDB ORM
  - ▸ mongoose tutorial
  - ▸ mongoose tutorial #2

```
1  app.get('/', function(req, res){
2     res.send('Hello World');
3  });
4
5  app.listen(3000);
6  console.log('Listening on port 3000');
```

AngularJS

- AngularJS
  - front-end code
    - extends HTML with Javascript
    - adds routing, controllers, models, views on the client side
  - Screencast
  - start with angular seed
    - angular express seed
    - angular express mongoose blog

Ember

▸ ember

- front-end code
  - uses Handlebars templates, similar to Jinja (Python) and erb (Ruby)
  - adds routing, controllers, models, views on the client side
  - has conventions, like Rails – name and structure given by ember
- ▸ Screencast
- ▸ node.js, express and mongoose

Other Resources

- ▸ Todo MVC example todo app in different frameworks
- ▸ Travis CI continuous integration service for GitHub projects