Server Side Languages 2

Frameworks

MVC

- https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller
- Divides a given application in three parts.
- Separates 'internal representations of information'! What does this mean?
- Allows for code reuse. How?
- Allows for parallel development. How?

Server Side Web Frameworks

MVC

- Obviously!

Model

- How is the data represented.
- Usually involves defining columns and their values.
  - model: User
    - full_name: string, 100
    - username: string, 30, not null, unique
    - email: string, 100, not null, primary key
    - password: string, 256
    - signup: timestamp
- Involves abstraction
  - Object Relational Mapping (ORM)
  - Creates an abstraction of what the database looks like.
  - Can add additional behavior to the model.
- Database Independent.
  - Underlying database can change without affecting the abstraction.
- Automation!
  - Schema created for you.
  - Queries use programming language instead of RAW queries.

- Validation
  - Validate fields based on definition.
  - Example checks.
    - Is field correct length.
    - Is field that is required filled in.
    - Is field type correct (is email entered an email).

Example SQL Model:

```
CREATE TABLE user (
 id int not null auto_increment,
 full_name varchar(100),
 username varchar(30) unique,
 email varchar(100) primary key,
 password varchar(256),
 signup timestamp
 );

 SELECT * FROM user WHERE username = 'netid';
```

Example Model Abstraction

```
from django.db import models
from datetime import datetime

class User(model.Model):
  id = models.AutoField()
  full_name = models.CharField(max_length=30)
  username = models.CharField(max_length=30, required=True)
  email = models.EmailField(max_length=100, required=True, primary_key=True)
  password = models.CharField(max_length=256)
  signup = models.DateTimeField(default=datetime.now())

user = User.objects.get(username='netid')
```

Views

- Involves rendering of HTML.
- HTML TEMPLATING
- https://docs.djangoproject.com/en/1.11/ref/templates/builtins/

Example RAW Python

```
data = '<html><head><title>' + title + '</title></head><body><p class="paragraph">' + text + ' by <strong>' ...

print (data)
```

Example Django Template

```
<html>
  <head>
    <title>{% title %}</title>
  </head>
  <body>
    <p class="paragraph">{% text %} by <strong>{% user.full_name %}</strong></p>
  </body>
</html>
```

API (and REST)

- API: Application programming interface.
  - https://en.wikipedia.org/wiki/Application_programming_interface
  - Set of subroutines definitions, protocols (rules) and tools.
  - A way of interacting with a Resource that we don't necessarily have access to or control of.
  - Usually for developers.
  - Libraries.
  - Operating Systems.
  - Web APIs
    - Code Reuse.
      - Same 'data' interface for Web App and Mobile App.
    - Third party applications.
      - Add a feature that is not available.
      - Addons/Games.
      - Enhance your own applications.
    - Usually presented as a RESTful Interface.
      - Stateless!
      - Most of the time requires permission.

        - Operations determined by HTTP verbs.
            - GET
            - POST
            - PUT
            - DELETE
- Involves returning data to user.
    - JSON
    - XML
    - HTML (In which cases?)
- Several uses:
    - Front-End.
    - Single Page APPs.
    - Third Party Applications.

Controllers

- Business Logic.
- Database interaction.
- Authentication checks.
- Permission checks.

Middleware (Addons)

- Tools added to enhance MVC.
- Tightly integrated to the Framework.
- Example
    - HTML Rendering tag.
    - Caching scheme.
    - Permission checks.

URLs

- Mapping a URL to a controller.
- Clean URLs.

STACK

- Usually a combination of several technologies.
- Database + Server Side Framework + Client Side Framework!

Examples

- Python
  - **Django**
  - Web2Py
  - Flask
  - ...
- NodeJS
  - **Express**
  - Sails (API)
  - Nodal
  - ...
- PHP
  - Laravel
  - Phalcon
  - Symfony
  - Yii
  - Zend
  - Codeigniter
  - CakePHP