CSCI 1320
Creating Modern Web Applications
Lecture 20: Newer Server Technologies
Frameworks

- Web applications are a lot of work
  - Much of it is busy work
- Routine chores and programming
  - Administration
  - URL -> command mapping
  - Providing HTML pages
  - Data structure mapping
- Frameworks try to simplify this
Model-View-Controller

Web Browser  Web Server  Database

Data Display  Data Management  Data Store
Model-View-Controller

Controller

Data Management

View

Data Display

Model

Data Store
Model-View-Controller

- Basic idea is to separate the display, the data, and the logic
  - Each can be changed independently of the others
- Exactly how this is done varies from case to case
  - Some do it with a common data abstraction
  - Some do it with callbacks
  - All call themselves MVC
- Different people mean different things
DRY Principle

• Don’t Repeat Yourself
  • Every piece of knowledge must have a single unambiguous authoritative representation within a system

• Why have 3 different representations of the data
  • More code to maintain
  • More code to change when data changes
  • More chance for bugs
Django and Ruby on Rails

• Widely used
  • Django: instagram, pinterest, ...
  • Ruby/rails: github, basecamp, ...
• Similar frameworks exist (e.g. Flask)
• Mostly a back end technology
  • Can be paired with a templating engine
• Require knowing python/ruby
  • In addition to JavaScript, HTML, ...
Django/Ruby Frameworks

- Express-like dispatch
  - Based on static tables, not executed code
  - With functions to handle the results
- Logic to control deployment, server setup, etc.
- Libraries to handle common web app features
- Simple connection to database
- Simplified Data Management
DJANGO and Ruby/Rails

- Map from internal objects to SQL automatically
  - Changes in the object -> SQL updates
  - Objects created automatically from SQL database
  - SQL Tables created automatically from object definition
  - Changes to object definition change the database
- Map from internal objects to HTML automatically
  - Using templates
- Map from internal objects to JSON automatically
  - Changes in the object -> go to web site if needed
- OBJECT-RELATIONAL MODELING
Object-Relational Modeling

• Not limited to Django-Ruby
  • There are libraries to provide some of this functionality
• Not limited to SQL back ends
  • NoSQL databases can be used as
    • Direct mapping to object from json
  • Cache the current state in memory as objects
    • This allows fast query at times
  • Update updates memory and the database
• What is the problem with this?
RESTful Web Applications

- Client-Server model
  - Client handles presentation, server handles storage
- Stateless
  - Add data needed for request is passed
- Client maintains data
  - Sends updates, requests to server
  - Using commands encoded in URL
RESTful API HTTP Methods

- **Collection API** .../collection
  - **GET**: return list of elements in the collection
  - **PUT**: Replace the entire collection
  - **POST**: Add an entry to the collection
  - **DELETE**: Delete the entire collection

- **Element API**: .../collection/:item
  - **GET**: Retrieve the given item
  - **PUT**: Replace or create the given item
  - **DELETE**: delete the given item

- **Action API**: .../collection/:item/verb
DJANGO/RUBY with REST

- **URL identifies the object in the server**
  - What field to access or change
  - New value of the field (using PUT)
- **Front end makes changes to long term objects**
  - By sending POST requests
- **Front end gets current state of objects**
  - By sending GET requests
AngularJS

- Templates that are executed at run time
- Automatically update the page as values change
- MVC (Model-View-Controller)
  - Model = the data structures
  - View = the template
  - Control = commands that modify the data
- Combine this with Object-Relational Modeling
  - Make a simple, consistent web application
React-JS

- Templates mixed with JavaScript code
  - Expressed as functions
  - With HTML
  - And embedded code
- Can be done either server side or client side
  - Use for templating in the server
Content Management Systems

- Creation and Modification of digital content
  - The contents of the web site
- Easy to create good-looking sites
  - With modern bells and whistles (e.g. slide shows)
- Easy to update the contents
  - For a non-programmer
- Standard interaction mechanisms often included
  - User accounts, ...
  - Blogs, Wikis, ...
Content Management Systems

- **WordPress**
  - The standard
  - PHP based
  - Extensible with modules or your own code
- **Drupal**
  - Relatively common, more flexible
  - Fewer modules and features
  - PHP Based
- **Django-CMS**
  - Used for Brown CS web site
  - Lots of others available
CMS Features

• Templating engine
  • MVC model – separate presentation from application logic
  • Reusable pieces
• Roles and permissions
  • Authentication
  • Roles: admin, author, editor, user, ...
  • Hide complexity
CMS Features

- **In-Browser Editing**
  - Either separate editor on on-page editing
  - Layout and style
  - Images and media
  - Plugins such as Google maps

- **Publishing workflow**
  - Create -> Edit -> Approve -> Publish -> Update -> Approve ...

- **Versioning**
  - Revert, record of who did what
CMS Features

• Multilingual
  • Support for different languages
• Accessibility support
• Multi-site
  • Multiple sites running on one server
• Tree-like page structure
  • With appropriate permissions
• RESTful URLs
• Analytics
CMS Integration

- Can use CMS as a part of the web site
  - For the appropriate pages
- Code the other pages separately
  - Node.JS or other front end
- Integration in various ways
  - Django with Django-CMS
  - Reverse Proxy
    - Front end server redirects to appropriate back end
Next Time

• Wednesday: HCI Introduction and Web Site Navigation
• Friday: HCI Design
• Monday/Wednesday: Poster Sessions (Sayles)
• Friday: Design lab
• Spring Break
JavaScript

• JavaScript was designed for writing simple things
  • Short routines to check inputs
  • Short routines to provide interactivity
• Not necessarily the best language for large systems
  • No name spaces or module structure
  • Prototype-based object model
  • Easy to write erroneous programs
    • No compile time
    • No type checking
  • Improvements in Node.js, not in the front end
• jQuery, angular, and other libraries help
  • But really aren’t enough