# Web Applications

# CSE183

Fall 2020

## User Interfaces

Baskin Engineering
UC SANTA CRUZ

## Notices

- **Assignment 4** due 23:59 **Thursday October 29**
- **Quiz 2** during class **Friday October 30**





## Today's Lecture

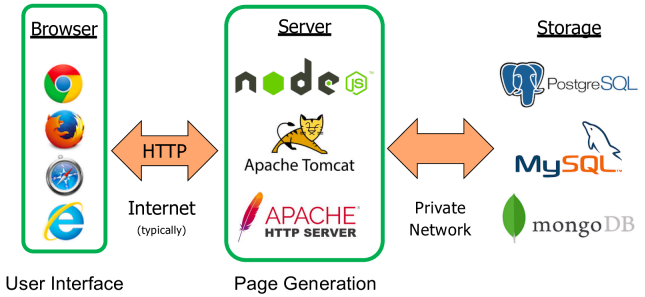
- DOM Events Revisited
- Event Concurrency
- 1st, 2nd, 3rd Generation Web Application Frameworks
- Templating
- Model View Controller
- 4th Generation Web Application Frameworks

- Assignments 2 & 3 Review
- Assignment 4 Introduction
- Questions





## Full Stack Web Applications



## DOM Events Revisited

- Handlers

```
<div onclick = "alert('Hello World');">Hello World</div>
```

element event JavaScript

Or

```
let mouseClick = function(event) { alert('Hello World'); }
element.addEventListener('click', mouseClick);
```

- The `Event` object and descendants `MouseEvent`, `KeyboardEvent`, etc.
- Precedence
- Capture and Bubbling
- Timer Events




## Event Concurrency

- Events are serialized and processed one-by-one
- Event handling *does not* interleave with other JavaScript
  - Handlers run to completion
  - Not preemptable
  - Not multi-threaded
- Makes reasoning about concurrency easier ☺
  - Rarely need locks
- Background processing is much harder than with threads ☹




## Event Based Programming

- Must wait for something to invoke your code
- Must return quickly from the handler
  - Otherwise Web App becomes unresponsive
- Key is to maintain control through events
  - Make sure you have declared enough handlers
  - Timers should be used only as a last resort
- Node.js provides an event dispatching mechanism for server-side JavaScript programming
  - We'll see this later in the class




## Web App History Revisited

- Initially static HTML files with HTML forms for input
- Then Common Gateway Interface (CGI)
  - Some URLs map to executable code that generates HTML
  - Program exits when HTML has been generated
  - Stateless servers:
    - Requests are independent
    - No state preserved between executions
  - Perl was a popular language for CGI applications
    - General purpose interpreted language, based on C

## 1st Generation Web App Frameworks

- Languages:
  - PHP
  - ASP ( Microsoft - Application Server Pages )
  - Java Servlets
- Language runtime embedded in Web Server
- Frequently used templates to mix code and HTML / CSS
- Needed web-specific language extensions / packages
  - URL Handling
  - HTML generation
  - Stateful Sessions
  - Database Interaction
  - Etc.




## 2nd Generation Web App Frameworks

- Examples:
  - Ruby on Rails
  - Django
- Page generation still in Web Server
- Encouraged Model View Controller decomposition
  - We'll look at this later as it's still heavily used
- Typically supported object-relational mapping (ORM)
  - Simplifies database integration by exposing database contents (tables and rows) as classes and objects
  - Made generating dynamic pages easier



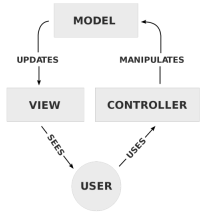

## 3rd Generation Web App Frameworks

- Example:
  - AngularJS
- JavaScript frameworks running in browser
  - No server-side page generation
- Frameworks now independent of server-side capabilities
  - Can write once and run in any (or at least, many) servers
- Support good ideas / concepts from earlier generations
  - Model View Controller
  - Templating
  - Separation of Style and Content




## Model View Controller (MVC)

- A mature software design pattern for user interfaces
  - Originated in the 1970's
- **Model**
  - Takes care of the application's data
    - JavaScript objects
- **View**
  - Defines what the web page looks like
    - HTML & CSS
- **Controller**
  - Manipulates the model which causes changes in the view
    - JavaScript code





https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

## 3rd Generation Views

- However generated, browsers only render CSS styled HTML
  - This the fundamental principle of Web Applications
- Templates are a common page generation technique
  - Write HTML document with the parts of the page that are the same regardless of situation
  - Add tags indicating where sections of the page need to be changed for specific situations
  - The templating system replaces the tags with information relevant to the current situation
- Benefits of templates ( compared to modifying DOM via JavaScript )
  - Easy to visualise document structure
    - Tagged HTML will still render in browser
  - Templating can occur in the server or in the browser




## Template Example - Angular

- Angular has a rich templating syntax
  - Loops, conditionals, subroutines, …
- Simple example:

```
<body>
  <div class="greetings">
    Welcome back {{models.user.fname}}!,
  </div>
  <div class="applicationcount">
    You have {{models.application.count}}
      applications to approve.
  </div>
</body>
```




## 3rd Generation Controllers

- JavaScript running in browser
- Connect models and views
  - Communicate with server to get models and send updates
- Organise view templates
  - Control which ones are being shown
- Handle user interactions
  - Button clicks, menu activations, text entry, etc.




## Controller Example - Angular

```
function userLandingView ($scope, $modelService) {
  $scope.models = {};
  $scope.models.users = $modelService.fetch("users");
  $scope.models.applications = $modelService.fetch("applications");
  $scope.okPushed = function okPushed() {
    // Code to execute when OK button is pushed
  }
}
```

`$scope` is the link between Angular Controller and View


https://docs.angularjs.org/guide/scope

## 3rd Generation Models

- All dynamic (i.e. non-static) information needed by the view templates or controllers
- Traditionally tied to application's database schema
  - In ORM a model is a row in a table
- Web application's model data needs are specified by the view designers but need to be persisted by the database
- Problem:
  - Traditional relational database schemas don't like changing
  - Web Application model data frequently changes
    - e.g. *"user will obviously like this view better if we add a A and get rid of B"*




## Model Example - Angular

- Angular provides support for fetching data from the server
  - REST APIs ( we'll see these later )
  - JSON is a common on-wire data representation
- Server representation is "whatever you like"
  - Angular is agnostic - it only cares about the on-wire format
  - e.g. Mongoose Object Definition Language (ODL) for MongoDB:

```
var userSchema = new Schema({
  firstName: String,
  lastName: String,
});
var User = mongoose.model('User', userSchema);
```




## 4th Generation Web App Frameworks

- Examples:
  - **React**, Vue, Angular 2
- Adopted techniques and concepts from earlier generations:
  - Browser side JavaScript
  - Model View Controller
  - Templating
- Concentrate on JavaScript components rather than HTML
- Write into a server-side Virtual DOM
  - When all modifications complete DOM replaced with contents of VDOM in a single operation ☺
    - Compare to tens of thousands of individual DOM modifications ☹




## Assignment 2 - Review

- The browser is in charge
  - Ask for anything you like, but in the end the browser decides what you get ☹
- CSS is like any other programming language
  - You can write sloppy CSS, or you can write tight CSS
  - CSS has re-use capabilities

```
.letter {
  width: 60px;
  height: 60px;
  text-align: center;
  background-color: yellow;
  color: black;
}

.number {
  width: 60px;
  height: 60px;
  text-align: center;
  background-color: yellow;
  color: black;
}
```

☹

```
.letter, .number {
  width: 60px;
  height: 60px;
  text-align: center;
  background-color: yellow;
  color: black;
}
```

☺

# Assignment 3 - Review

- Don't re-invent the wheel
  - Use Regular Expressions rather than write the logic yourself
- Testing is good
  - But be careful, easy to worry about edge cases that were not required
- Code Coverage

```
if (something) {
   // whatever
}
```

the `if` introduces a branch, both of which need to be covered

- Conflicts between coverage and linting

```
for (const prop in map) {
  if (map.hasOwnProperty(prop)) {
   // use prop
  }
  if (map.hasOwnProperty(prop) && prop != '')) {
   // use prop
  }
```

Google rules say we need to check prop before using it

Whoops! Uncovered branch ⊗

Branch now covered ☺

```
test('Undefined Tag', () => {
  const t = new Templater('Mary {{had}} a {{little}} {{lamb}}');
  expect(t.apply({'': ''})).toBe('Mary a');
});
```





# Assignment 4 - Introduction

- Use everything we've looked at so far ☺
  - HTML, CSS, JSON, JavaScript, DOM, Events ( woo-hoo! )
- It's significantly more work than Assignment 3 ☹
  - But not impossible, so take your time and keep calm
  - Get started **TODAY!**
- Three Components:
  - Basic: 3 points
    - Fill in an HTML table from JSON
  - Advanced: 2 points ( 1 for functionality 1 for look )
    - Write a simple date Picker
  - Stretch 2 points ( 1 for functionality 1 for look )
    - Write a (slightly) more sophisticated date Spinner
- You may want to write your own tests for Advanced & Stretch





# Assignment 4 - Basic

| {{H1}} | {{H2}} | {{H3}} |
|---|---|---|
| {{R11}} | {{R12}} | {{R13}} |
| {{R21}} | {{R22}} | {{R23}} |
| {{R31}} | {{R32}} | {{R33}} |
| {{R41}} | {{R42}} | {{R43}} |
| {{R51}} | {{R52}} | {{R53}} |

```
{
  "H1":"Name",
  "H2":"Address",
  "H3":"Age",
  "R31":"Billy",
  "R22":"14a Main st.",
  "R53":"32"
}
```

By Tag | By Id

Click Either Button

| Name | Address | Age |
|---|---|---|
| Billy | 14a Main st. | 32 |

```
{
  "H1":"Name",
  "H2":"Address",
  "H3":"Age",
  "R31":"Billy",
  "R22":"14a Main st.",
  "R53":"32"
}
```

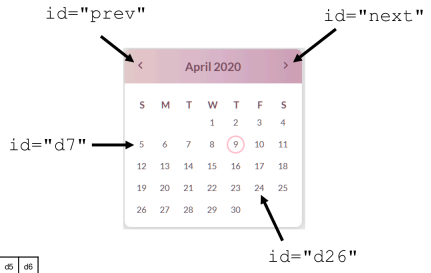
By Tag | By Id

After first click, only "By Id" will do anything



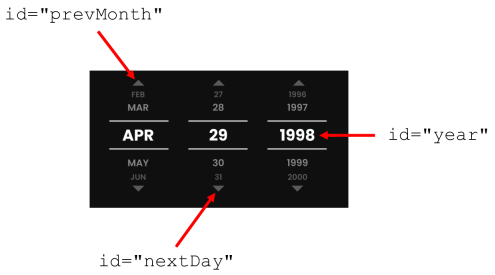


# Assignment 4 - Advanced - "Picker"




| d0 | d1 | d2 | d3 | d4 | d5 | d6 |
|---|---|---|---|---|---|---|
| d7 | d8 | d9 | d10 | d11 | d12 | d13 |
| d14 | d15 | d16 | d17 | d18 | d19 | d20 |
| d21 | d22 | d23 | d24 | d25 | d26 | d27 |
| d28 | d29 | d30 | d31 | d32 | d33 | d34 |
| d35 | d36 | d37 | d38 | d39 | d40 | d41 |

## Assignment 4 - Stretch - “Spinner”





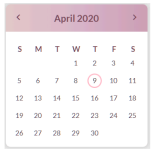



## Assignment 4 - Advanced & Stretch

- Work on the functionality ***first***

<
October 2020
>
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

- Then work on what they look like
  - Separate style from content ☺








## Upcoming Lectures

- Monday: Introduction to React
- Wednesday: Single Page Web Applications
- Friday: Quiz 2

## Tasks

- **Assignment 4** due 23:59 **Thursday October 29**