Input and Validation

Mendel Rosenblum
Early web app input: HTTP form tag

```html
<form action="/product/update" method="post">
  Product: <input type="text" name="product"/>
  Deluxe: <input type="checkbox" name="delux"/>
  <input type="submit" value="Submit"/>
</form>

- method="get" - Encode form properties as query params
  HTTP GET product/update?product=foobar&delux=on

- method="post" - Encode form properties as query params in message body
  HTTP POST product/update
  Content-Type: application/x-www-form-urlencoded
  product=foobar&delux=on
```
Rails input pattern using form POST

- **GET Page containing form**
  - Contains a method="post" form to a POST Page

- **POST Page - Validate and perform operation (typically create or update)**
  - If successful, redirect to a "done "page (possibly another GET Page) if successful
  - If failed validation, redirect page to the GET Page with incorrect fields highlighted
  - If error, redirect to some oops page

- **Validation needs:**
  - Protect integrity of storage (required fields, organization, security, etc.)
    - Need to enforce at server API
  - Provide a good user experience
    - Push validation closer to the user is helpful
Validation with AngularJS

- Rule #1: Still need server-side validation to protect storage system integrity
- Rule #2: Let user know about validity problems as early as possible
- Angular reuses the HTML form tag
  
  ```html
  <form name="myForm">
    <input type="text" name="myName" ng-model="name" required
           ng-minlength="3" ng-maxlength="20" />
  </form>
  
  Generates a scope object property under form name (myForm)
  $scope.myForm.myName has validation information
  ```
Angular validation information

$scope.myForm.myName

Status: $untouched, $touched, $pristine, $dirty, $valid, $invalid

Error: $error.required
  .minlength
  .maxlength

- Also updates classes on input tag (e.g. ng-invalid-maxlength)
- Can provide instant feedback on errors
Angular Material: md-input-container pattern

<form name="userForm" ...>

<md-input-container>
  <label>Last Name</label>
  <input name="lastName" ng-model="lastName" required md-maxlength="10" minlength="4">
  <div ng-messages="userForm.lastName.$error" ng-show="userForm.lastName.$dirty">
    <div ng-message="required">This is required!</div>
    <div ng-message="md-maxlength">That's too long!</div>
    <div ng-message="minlength">That's too short!</div>
  </div>
</md-input-container>
</form>
Asynchronous validation

- Can in background communicate with web server to validate input
  - Example: user name already taken
- Example: states search with md-autocomplete

```html
<md-autocomplete md-selected-item="ctrl1.selectedItem"
  md-search-text="ctrl1.searchText"
  md-items="item in ctrl1.querySearch(ctrl1.searchText)"
  md-item-text="item.display" placeholder="What is your favorite US state?">
  <span md-highlight-text="ctrl1.searchText">{{item.display}}</span>
</md-autocomplete>
```

- Trend towards using recommendation systems for input guidance
Single Page App Input

- Rather than POST with redirect you can do a XMLHttpRequest POST/PUT
- Angular supports two interfaces to XMLHttpRequest ($http and $resource)

```javascript
function FetchModel(url, doneCallback) {
  $http.get(url).then(function(response) {
    var ok = (response.status === 200);
    doneCallback(ok ? response.data : undefined);
  }, function(response) {
    doneCallback(undefined);
  });
}
```
Uploading models using $http.post

```javascript
$http.post(url, modelObj).then(function successCallback(response) {
    // response.status  --- HTTP status code
    // response.data     --- POST response if successful (decoded)
    // response.headers  --- HTTP response headers
}, function errorCallback(response) {
    // Network Error case (webServer or network down?)
})
```

- App must wait for reply since errors may occur on server
  - Need some user interface way of communicating this to the user
$resource - RESTful server access

- In REST APIs you have resources named as URLs

```
var resource = $resource(resourceURLTemplate, paramDefaults);
```

- And operations on resources:

```
resource.get(params, doneCback) - {method:'GET'}
resource.save(params, doneCback) - {method:'POST'},
resource.query(params, doneCback) - {method:'GET', isArray: true}
resource.remove(params, doneCback) - {method:'DELETE'},
resource.delete(params, doneCback) - {method:'DELETE'}
```
var testRes = $resource("/test/info");
var infoModel = testRes.get({}, function () {
    console.log('infoModel', infoModel);
}, function errorHandling(err) {
    // Any error or non-OK status
});

var userRes = $resource("/user");
uuserRes.save({user: 'mendel', password: 'pwd'}, function () {
    // Success
}, function errorHandling(err) {
    // Any error or non-OK status
});
Server-side validation

- Regardless of validation in browser server needs to check everything
  - Easy to directly access server API bypassing all browser validation checks

- Mongoose allows validator functions

```javascript
var userSchema = new Schema({
  phone: { type: String,
    validate: {
      validator: function(v) {
        return /d{3}-d{3}-d{4}/.test(v);
      },
      message: '{VALUE} is not a valid phone number!' 
    }
  }
});
```
Some integrity enforcement requires special code

- Maintaining relationship between objects
- Resource quotas