State Management

Mendel Rosenblum
Our small, read-only photo app is deceptively simple

- Model, View, Controller - All setup on startup and static
  - Can have a nice modular design of view components

- Add in Session State and object creation and updating
  - Things get more complex particularly for our single page app

- Examples:
  - Users logs out and logins into the app with a different login name
  - User add news comments or photos.
Session state

- Must be kept in sync between the browser app and the server
  - Who, if anyone, is logged in?

- Server will need to reject any requests from users not logged in
  - Model fetching done only at view/controller startup might not work

- Consider transitions of your photo app
  - Login - Not logged in to logged in
    - At app startup most models are not available (e.g. sidenav user list) but become available after login is completed.
  - Logout - Logged in to not logged in
    - Requests to web server that worked before will now fail
Models updates

- Consider what happens when new objects like users, photos, or comments are added.
  - Models change
- Controller fetching model only at startup might not work
- Consider photo app adding a photo or comment
  - Model refresh needed
Controllers are interested in outside events

- How to keep a modular design but allow controllers to be notified of things happening outside of it?
  - Example: a view component and an add component

Angular approach: **events**

- `$scope.$on(eventName, listener)`: Function listener is called when event is raised
- `$scope.$emit(eventName, args)`: Event is raised and goes up $scope chain
- `$scope.$broadcast(eventName, args)`: Event is raised and goes down $scope chain

- Frequently used pattern - $broadcast of event the $rootScope
  - Controller: `$scope.$on('photoUploaded', reloadModel);`
  - Photo upload dialog: `$rootScope.$broadcast('photoUploaded');`
Dealing with other model changes

What happens if another user adds a photo or comment? Options:

1. **Do nothing: Easy!**
   - User won't see new material until they do something that caused the model to be refreshed
   - Very disconcerting if they don't see their own changes

2. **Poll: Periodically check for changes or just refetch the model**
   - Can provide a UI widget to trigger model refresh

3. **Server push: Have the server push model changes as soon as they occur**
   - User sees updates as soon as possible
   - Might conflict with user changes or be disconcerting for the user
   - Implementation is easier with Web Sockets
Photo App with sessions and input

- App needs to track who (if anyone) is logged in
  - Ideally held in an Angular Service
  - OK to keep in the mainController's $scope that is a parent of all the view components

- Need to handle the no one logged-in case
  - Need to add code to controllers to handle this
  - Handling deep linking:
    ```javascript
    $rootScope.$on("$routeChangeStart", function(event, next, current) {
      if (noOneLoggedIn() && (next.templateUrl !== loginViewTemplate)) {
        // Force all views to the login view template
        $location.path("/login-register");
      }
    });
    ```

- Use events to single when controllers should refresh their models
  - Add/update controllers broadcast when changes occur