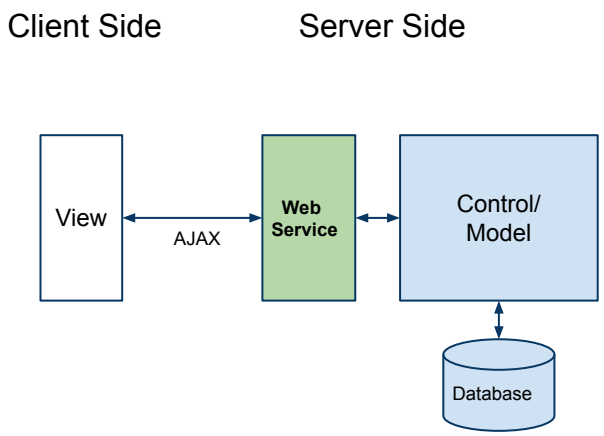
Web Applications 2017

Week 3, Slides 2

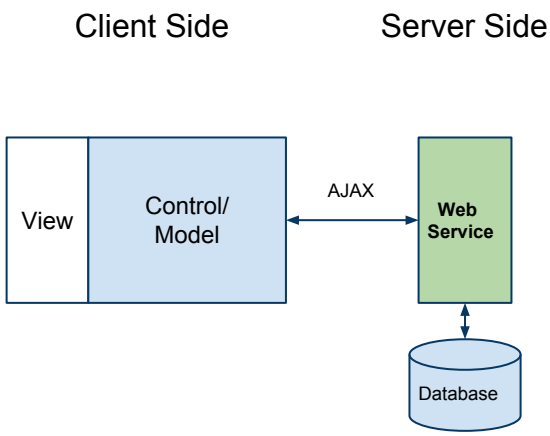
Content

- Client Side MVC
- Backbone
- Single Page Applications SPA
- Caching
- Authentication

REST Thin Client

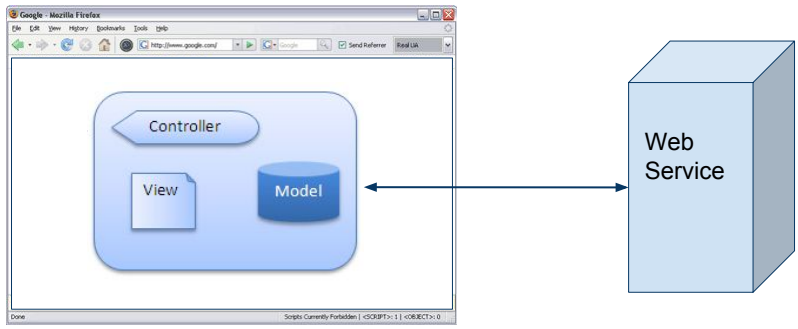


REST FAT Client



The topic for this lecture.

Client Side MVC



MVC “In house” or framework

Client Side Module System

```
// person.js
define(function() {

    function Person(id, name) {
        this.id = id;
        this.name = name;
    };
    return Person;
})

// personlist.js
define([
    '../service/personservice', 'models/person'
], function(service, Person) {
    'use strict';


})
```

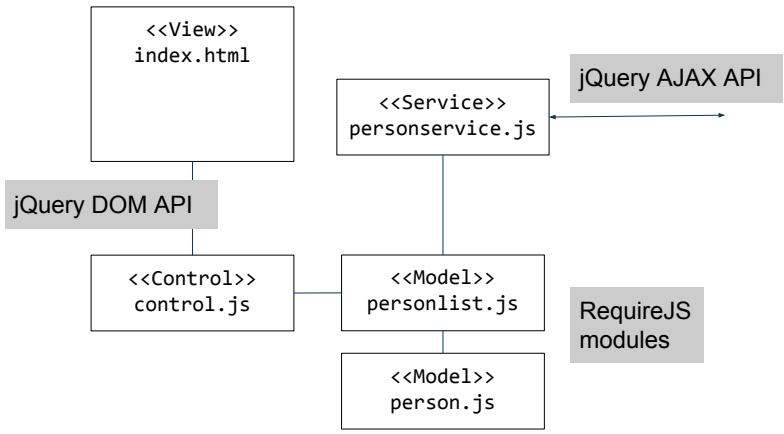
Dependencies (js files)

Injected objects (use in function)

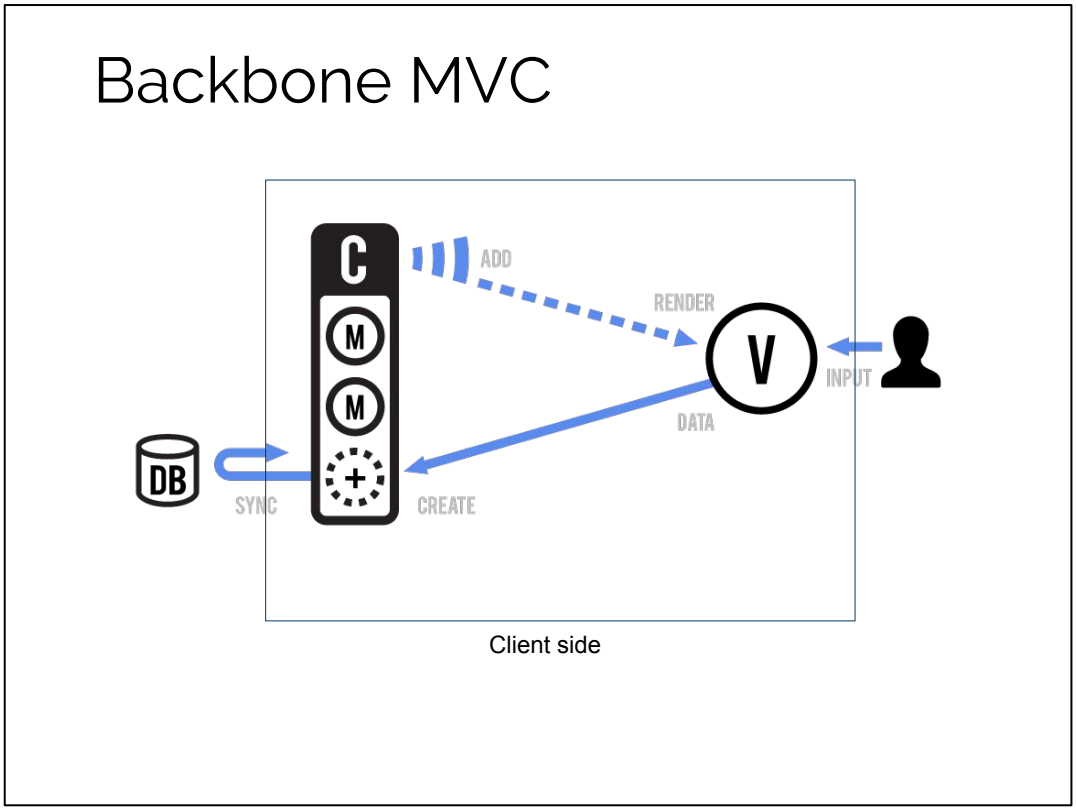
Readings

- RequireJS

In House MVC



Se also optimistic vs pessimistic AJAX



Backbone is a client side MVC framework.

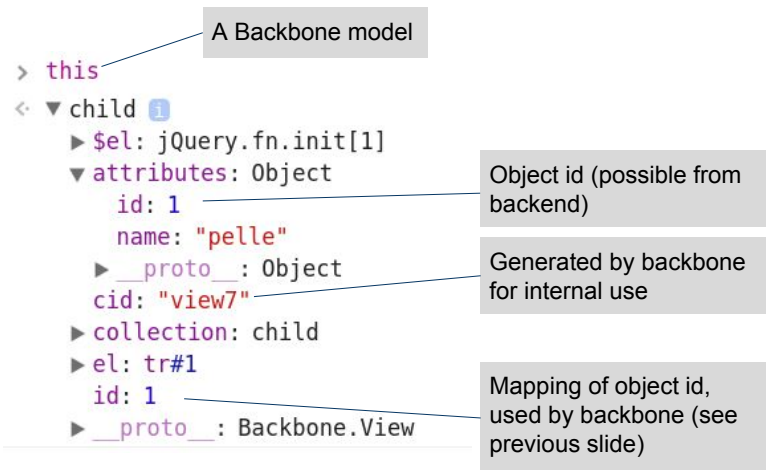
Readings

- Backbone.js
- Developing Backbone apps

Backbone Models

```
// A Model
var Person = Backbone.Model.extend({

    defaults: {
        name: 'noname'
    },

    idAttribute: "id",   // See next slide

    ...
});

var p1 = new Person(1, "pelle");
```

Backbone Model Id's



Backbone Events

// There are more ...
"add" (model, collection, options) — when a model is added to a collection.
"remove" (model, collection, options) — when a model is removed from a collection.
"update" (collection, options) — single event triggered after any number of models have been
"change" (model, options) — when a model's attributes have changed.
"change:[attribute]" (model, value, options) — when a specific attribute has been updated.
"destroy" (model, collection, options) — when a model is destroyed.
"all" — this special event fires for *any* triggered event, passing the event name as the first argument followed by all trigger arguments.

```
// Also possible: Use Backbone as eventbus
var object = {};
_.extend(object, Backbone.Events);  // _ = underscore library
object.on("alert", function(msg) {
  alert("Triggered " + msg);
});
object.trigger("alert", "an event");
```

-

Backbone Views

```
var PersonView = Backbone.View.extend({
    // Will create a new HTML element, referenced as "el"
    // Also: $el = $(view.el) i.e. el wrapped by jQuery
    tagName: "li",
    className: 'person', // optional
    id: 'person', // optional

    initialize: function(person) {
        this.person = person; // Data to display
    },
    // Template
    personTpl: _.template($('#item-template').html()),
    render: function() {
        // Use $el so we can call jquery methods
        this.$el.html(this.personTpl(this.person.attributes));
        return this;
    },
    // Connect listener (from GUI)
    events: {
       'click .delete': 'delete' // delete is a method
    },
});
```

Underscore templating, we'll use Mustache

-

Backbone Collections

```
var PersonList = Backbone.Collection.extend({
   model: Person, // A constructor

    initialize: function() {
        this.on('all', function(e) {
            console.log(e);
        });
    }
});


var personList = new PersonList();

// Built in methods
personList.add(new Person(2, "kalle"));
personList.remove(2);
```

-

Backbone REST

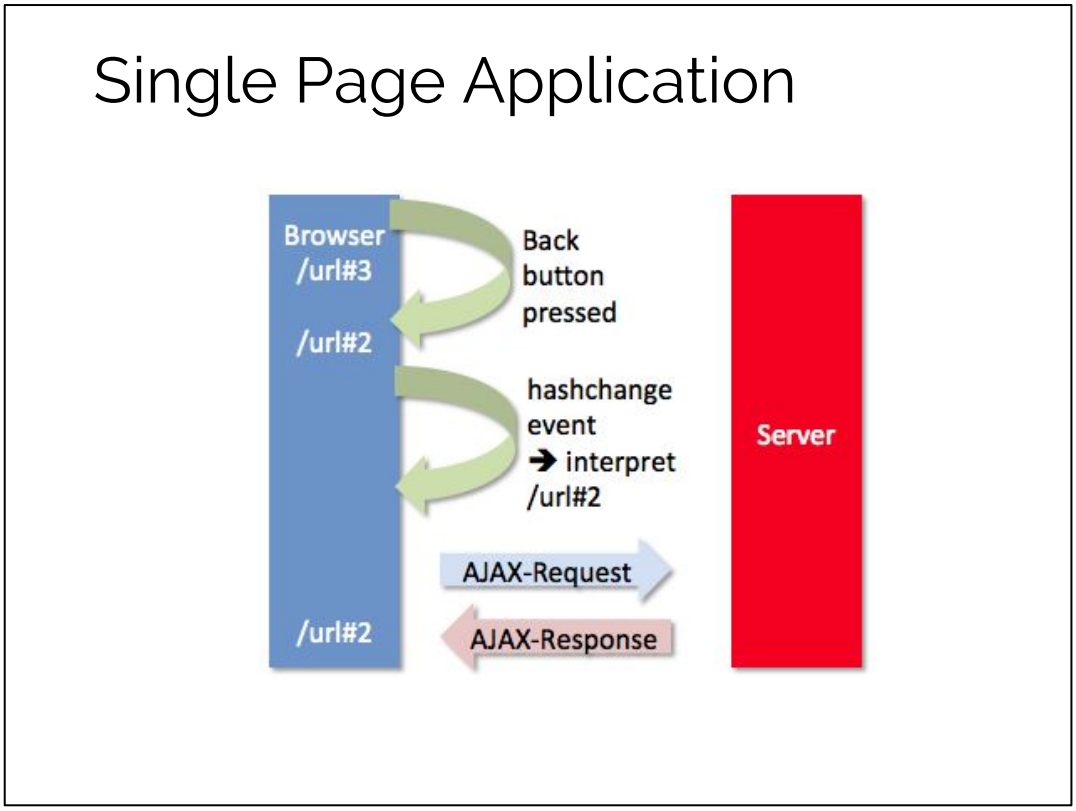
```
var PersonListREST = Backbone.Collection.extend({
    model: Person,
    url: 'http://localhost:3000/list',
    ...
});

var plr = new PersonListREST();
plr.fetch().done(function() {
    console.log(plr.length);
    console.log(plr.models);
}).fail(function(e) {
    console.log(e);
})

plr.create(otto, ...});

plr.fetch().done(function() {
    var p = plr.get(1);
    p.destroy();
});
```

-

Backbone + Mustache

```
define([ … 'text!templates/person.html'
], function(..., Mustache, personTemplate) {

    var PersonView = Backbone.View.extend({

        template: Mustache.parse(personTemplate),

        render: function() {
            var m = Mustache
               .render(personTemplate, {"person": this.person});
            this.$el.append(m);
        },


});
```

May use “any” templating system



Using AJAX we don't need any page loads, application stays on same page

- How to bookmark.
- Forward backward buttons.

Readings

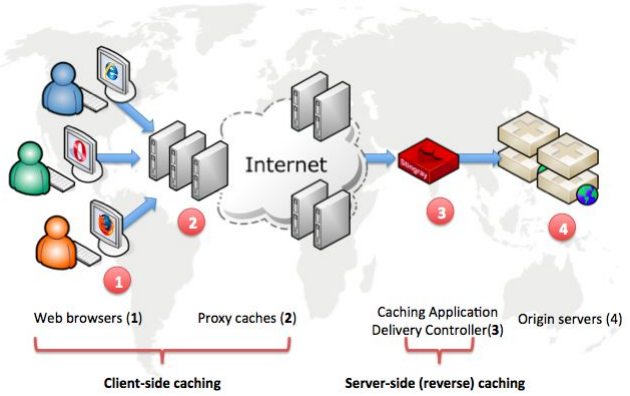
- Problems with bookmarks, forward/backward, favorites

Backbone Router

```
var Workspace = Backbone.Router.extend({

  routes: {
    "help":                 "help",    // #help
    "search/:query":        "search",  // #search/kiwis
    "search/:query/p:page": "search"   // #search/kiwis/p7
  },

  help: function() {
    ...
  },

  search: function(query, page) {
    ...
  }

});
```

Backbone.Router provides methods for routing client-side pages, and connecting them to actions and events.

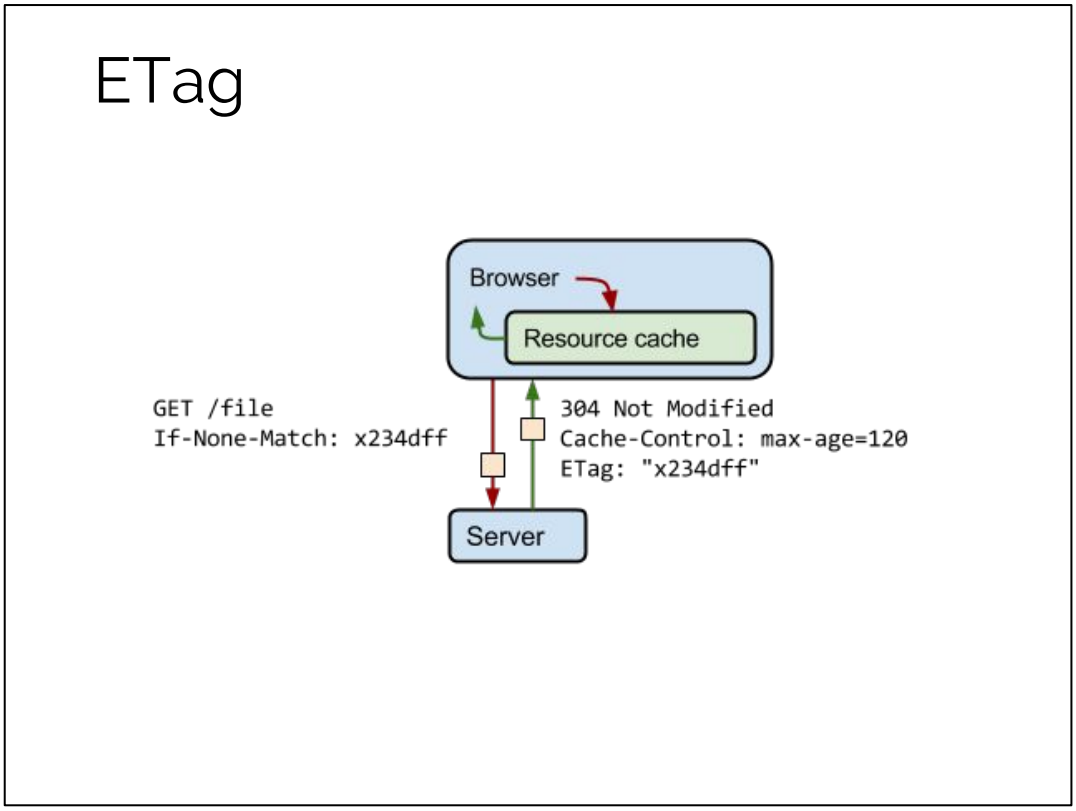
Caching



" … If, every time a browser processed a URI beginning http://..., it actually fired off a GET at a server, and if, every time a GET hit a server, the server actually recomputed and sent the data, the Web would melt down PDQ. There is a lot of machinery available, on both the client and server side, to detect when the work of computing results and sending them over the network can be avoided. Normally, we use the term "**caching**" to refer to all this stuff." // Tim Bray

Readings

- Digital Ocean Caching
- Caching Tutorial
- Google developers: Caching



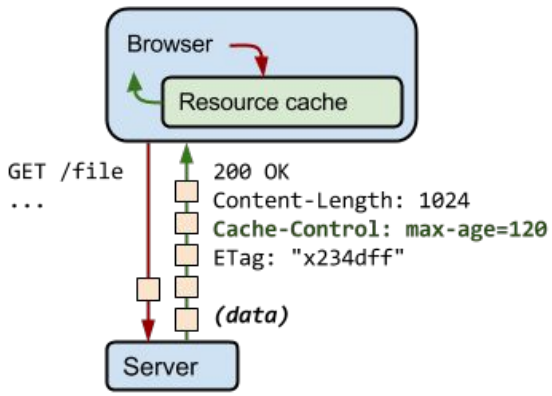
An ETag is an opaque identifier assigned by a web server to a specific version of a resource found at a URL.

- Example: ETag: "07cbbc409ec990f19c78c75bd1e06f215"

Readings

- ETag

Cache Control



Readings

- Cache-Control

Cache Control Header

```
HTTP/1.1 200 OK
Cache-Control: private, no-store, max-age=300
```

Some headers

Cache-Control:	**Description**
private	A cache mechanism may cache this page in a private cache and resend it only to a single client.
public	Shared caches, such as proxy servers, will cache pages with this setting. The cached page can be sent to any user.
no-cache	indicates that the returned response can't be used to satisfy a subsequent request to the same URL without first checking with the server if the response has changed
no-store	disallows the browser and all intermediate caches from storing any version of the returned response
max-age	How long resource is valid

Conditional Headers

If-Match
Succeeds if the ETag of the distant resource is equal to one listed in this header. By default, unless the etag is prefixed with 'W/', it performs a strong validation.

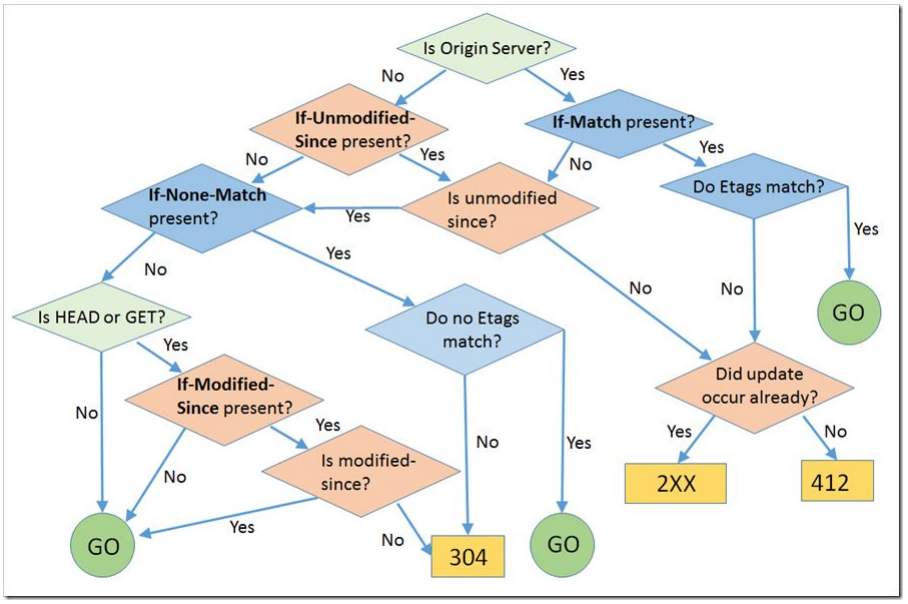
If-None-Match
Succeeds if the ETag of the distant resource is different to each listed in this header. By default, unless the etag is prefixed with 'W/', it performs a strong validation.

If-Modified-Since
Succeeds if the Last-Modified date of the distant resource is more recent than the one given in this header.

If-Unmodified-Since
Succeeds if the Last-Modified date of the distant resource is older or the same than the one given in this header.

If-Range
Similar to If-Match, or If-Unmodified-Since, but can have only one single etag, or one date. If it fails, a the range request fails and, instead of a 206 Partial Content response, a 200 OK is sent with the complete resource.

Conditional Requests



Readings

- [Conditional Requests]

Conditional GET

First Request

```
GET /rest_backend/webresources/cond/11 HTTP/1.1
...
Accept: */*

// Response
HTTP/1.1 200 OK
...
ETag: "053fde96fcc4b4ce72d7739202324cd49"
```

Second Request (conditional)

```
GET /rest_backend/webresources/cond/11 HTTP/1.1
If-None-Match: "053fde96fcc4b4ce72d7739202324cd49"

// Response
HTTP/1.1 304 Not Modified
ETag: "053fde96fcc4b4ce72d7739202324cd49"
```

Conditional GETs (this is latest JAX-RS, we do it a bit different)

- A conditional GET*) method requests that the entity be transferred only under the circumstances described by the conditional header field(s). The conditional GET method is intended to reduce unnecessary network usage by allowing cached entities to be refreshed without requiring multiple requests or transferring data already held by the client
- Headers are: If-Modified-Since, If-Unmodified-Since, If-Match, If-None-Match, or If-Range (If-* headers)

*) Also conditional updates (PUT)

-

Conditional PUT

First Request

```
GET /rest_backend/webresources/cond/11 HTTP/1.1
...
Accept: */*

// Response
HTTP/1.1 200 OK
...
ETag: "053fde96fcc4b4ce72d7739202324cd49"
```

Put Request (conditional)

```
PUT /rest_backend/webresources/cond/11 HTTP/1.1
If-Match: "053fde96fcXXX"    (other ETag value)

// Response
HTTP/1.1 412 Precondition Failed
```

If ETag on server doesn't match sent ETag (i.e. dirty data on client)

Authentication

In-house
- Let application handle

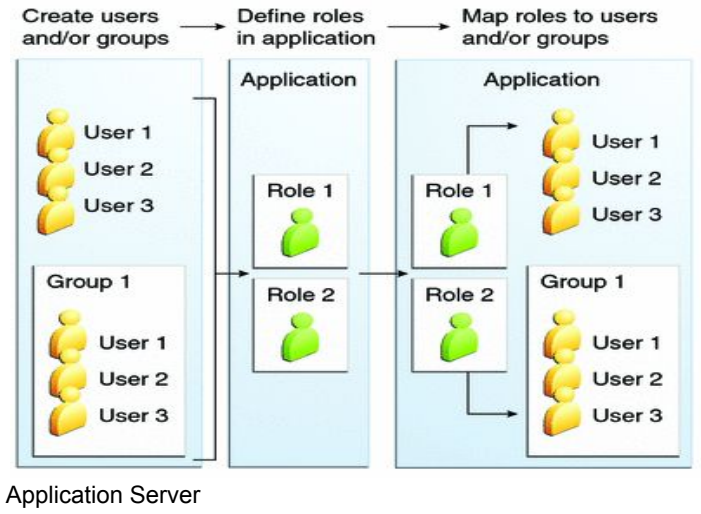
Node/Express
- Modules

JEE Standard
- Let application server handle

REST Application
- Hmm, stateless … no session …
- … use OAuth

Readings
- Passport (Express)
- Socialauth Java
- Google APIs
- JEE Security

JEE Authentication

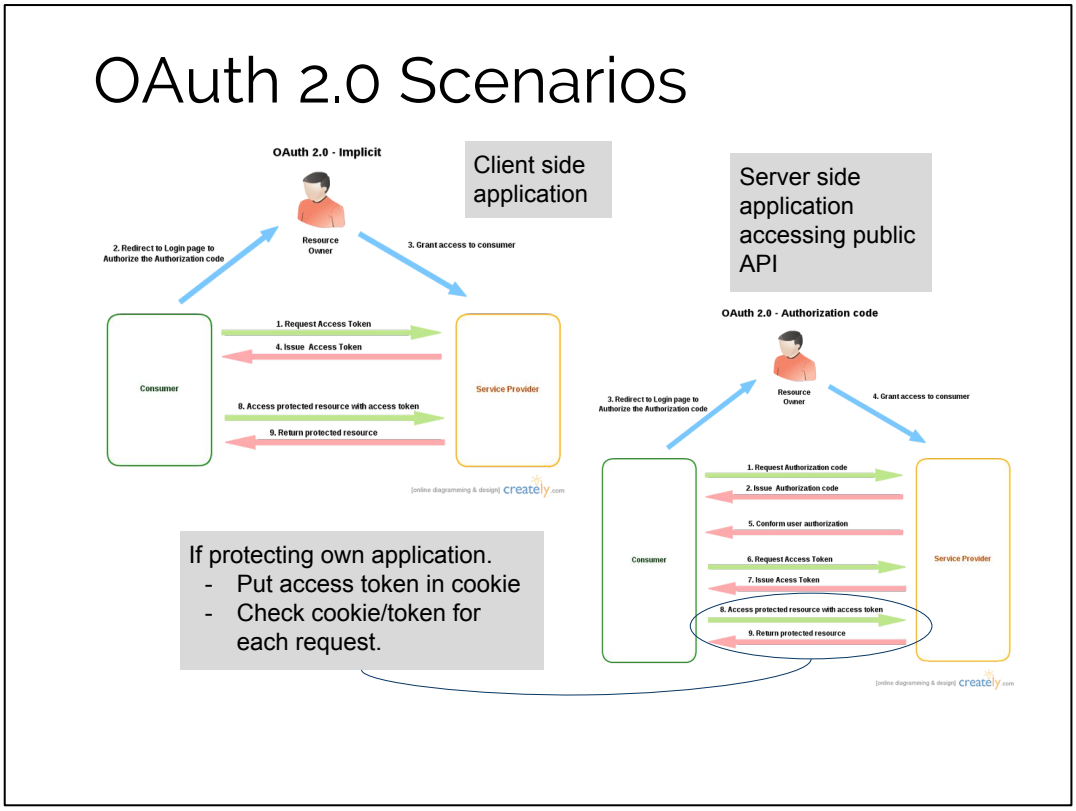


Standard JEE authorization technique, using realms

- A realm is a security policy domain defined for a web or application server. A realm contains a collection of users, who may or may not be assigned to a group
- Normally backed by database with users.

Readings

- Basic authentication
- Form-based authentication
- .. and more



REST should be stateless, can't store credentials (name/password) in session

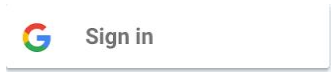
Solution:

- Send something (preferable time limited) representing the credentials with each request i.e. use OAuth
- There are version 1.0 and 2.0 not compatible

Readings

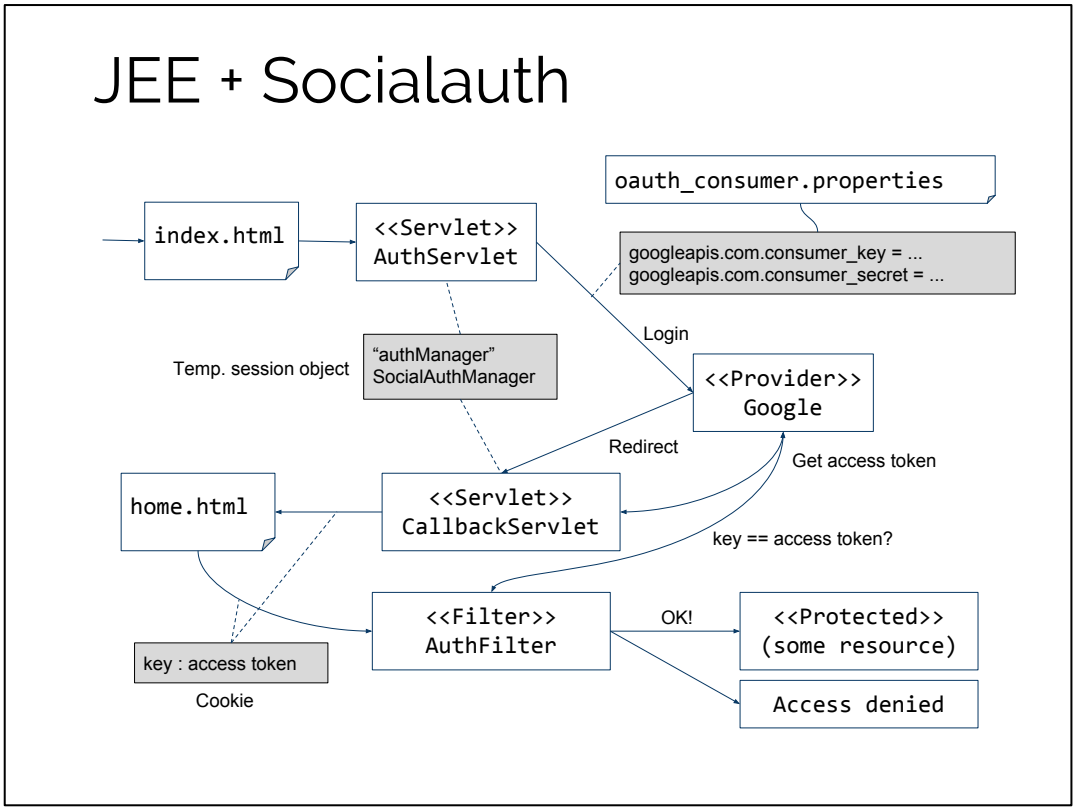
- Understanding OAuth2.0

Google Sign In




```
<div class="g-signin2" data-onsuccess="onSignIn"></div>

<script>
    function onSignIn(googleUser) {
        var profile = googleUser.getBasicProfile();
        console.log('ID: ' + profile.getId());
        console.log('Name: ' + profile.getName());
        console.log('Image URL: ' + profile.getImageUrl());
        console.log('Email: ' + profile.getEmail());
    }
</script>
```

Create a Google Developers Console project and client ID.



Application registred at Google API.

Possibly don't need to be as strict a sample:

- Use session object to store access token.