

5000th Anniversary

# Services + REST och OAuth

# Syftet med Services

Skapa ett Drupal API för att exponera webb-API:er.

# Officiella versionen

- Create a unified Drupal API for web services to be exposed in a variety of different server formats.
- Provide a service browser to be able to test methods.
- Allow distribution of API keys for developer access.

# Services

Funktionalitet uppdelad i tre olika sorters moduler som tillhandahåller:

- ♦ Servrar
- ♦ Tjänster
- ♦ Autentiseringsmekanismer (nytt i 2.x)

# Serverar

- ◆ REST
- ◆ XMLRPC
- ◆ JSONRPC
- ◆ SOAP
- ◆ AMF (Binary Flash RPC protocol)

# Tjänster

- ♦ Som exponerar core
  - ♦ Node, user, taxonomy, menu, file,
- ♦ Som exponerar andra moduler
  - ♦ Views
- ♦ Plus tjänster implementerade i andra contrib-moduler som jag inte kommer ihåg just nu.

# Autentisering

- ♦ OAuth
- ♦ Key authentication

# Att implementera tjänster

- ♦ Antigen som metoder
- ♦ ...eller (sen version 2.x) som resurser

# Metoder

- ✦ Tämligen likt menysystemet
- ✦ Varje tjänstemodul returnerar en icke-associativ array med metoder
- ✦ Metod-definitioner innehåller ett method-attribut: “node.get”, “node.view”, “node.save”, “node.delete”
- ✦ ...och information om callbacks, parametrar, åtkomstregler med mera.

```
<?php
/**
 * Implementation of hook_service().
 */
function node_service_service() {
  return array(
    // node.get
    array(
      '#method'           => 'node.get',
      '#callback'         => 'node_service_get',
      '#access callback'  => 'node_service_get_access',
      '#file'             => array('file' => 'inc', 'module' => 'node_service'),
      '#args'             => array(
        array(
          '#name'         => 'nid',
          '#type'         => 'int',
          '#description'  => t('A node ID.')),
          ...
        ),
      '#return'          => 'struct',
      '#help'            => t('Returns a node data.')
    ),
  ),
}
```

# Nackdelar

- ✦ Ingen semantik
  - ✦ `node.view` behandlas likadant som `node.delete`
- ✦ Inkonsekvenser
  - ✦ “`taxonomy.saveTerm`”, “`node.save`”
  - ✦ “`node.view`”, “`user.get`”
- ✦ Svårt att förändra genom alter-hooks

# Resurser

- ✦ Lägger till semantik till metoderna
- ✦ Naturlig gruppering kring resurser
  - ✦ inga mer “taxonomy.saveTerm”
- ✦ Metoderna delas upp i CRUD-operationer, actions, targeted actions och relationships

# Struktur - CRUD

- ◆ Resurs
  - ◆ Create
  - ◆ Retrieve
  - ◆ Update
  - ◆ Delete
  - ◆ (Index)

# Utökning till CRUD

- ◆ Actions
  - ◆ Som statiska klassmetoder:  
`Node::publish_my_drafts()`
- ◆ Targeted actions
  - ◆ Som klassmetoder: `$node->publish()`
- ◆ Relationships
  - ◆ Som targeted actions fast för läsoperationer:  
`$node->get_comments()`

# Alla gamla tjänster kan uttryckas som resurser

- ✦ Direkt replikering genom att lägga till alla de gamla metoderna t.ex.:  
taxonomy.saveTerm, saveVocabulary, getTree, selectNodes som actions på resursen taxonomy.
- ✦ Eller ännu bättre, göra om dem till ordentliga resurser (vocabulary och term).

# OAuth

- ✦ Säkert protokoll for att undvika “the password anti-pattern”.
- ✦ En standard som är på stark frammarsch.
- ✦ Klientimplementationer tillgängliga för de flesta stora och små språk.
- ✦ Se <http://oauth.net/code>



# OAuth-flödet för användaren

- ✦ Påbörjar auktoriserings-processen i en tredjepartsapplikation (consumer). Skickas vidare till vår sajt (providern).
- ✦ Användaren loggar in på providern och får förfrågan om att auktorisera consumern.
- ✦ Användaren skickas tillbaka till consumern.  
Färdigt!

# Token-baserad säkerhet

- ✦ Tre tokens (key och secret) är involverade: consumer-token, request-token och access-token.
- ✦ Consumern använder sin consumer-token för att hämta en request-token.
- ✦ Användaren auktoriserar vår request-token
- ✦ Consumern använder sin request-token för att hämta en access-token.
- ✦ Consumern använder consumer+access-token för att komma åt skyddade resurser.

# REST servern

- ♦ REST är designat för att fungera så bra som möjligt med HTTP.
- ♦ Alla resurser åtkomliga via en url
  - ♦ Create: POST <http://example.com/node>
  - ♦ Retrieve: GET <http://example.com/node/123>
    - ♦ Index: GET <http://example.com/node>
  - ♦ Update: PUT <http://example.com/node/123>
  - ♦ Delete: DELETE <http://example.com/node/123>

# Utökningarna av CRUD

- ◆ Actions
  - ◆ POST  
[http://example.com/node/publish\\_my\\_drafts](http://example.com/node/publish_my_drafts)
- ◆ Targeted actions
  - ◆ POST  
<http://example.com/node/123/publish>
- ◆ Relationships
  - ◆ GET  
<http://example.com/node/123/comments>

# Flera svarsformat

- ✦ XMLRPC returnerar alltid XML, JSONRPC ger JSON, SOAP ger XML+cruft och så vidare.
- ✦ REST är tämligen format-agnostiskt och kan ge svar i olika format baserat på filändelser eller Accept-headers.
  - ✦ GET <http://example.com/node/123.json>
  - ✦ GET <http://example.com/node/123.xml>
  - ✦ GET <http://example.com/node/123.php>
- ✦ Andra moduler kan lägga till och ändra svarsformat genom `hook_rest_server_response_formatters_alter()`.

# Alla svarsformat ärver klassen RESTServerView

```
/**
 * Base class for all response format views
 */
abstract class RESTServerView {
    protected $model;
    protected $arguments;

    function __construct($model, $arguments=array()) {
        $this->model = $model;
        $this->arguments = $arguments;
    }

    public abstract function render();
}
```

# Mer avancerade svarsformat

- ✦ De svarsformat som kan inte kan hanteras genom enkel serialisering
- ✦ RSS, iCal, xCal med flera
- ✦ Formatet kan då kräva att metoden skall implementera en datamodell som fungerar som en adapter.

# Exempel från xCal

```
function xcal_..._formatters_alter(&$formatters) {  
  $formatters['xcal'] = array(  
    'model' => 'ResourceTimeFeedModel',  
    'mime types' => array('application/xcal+xml'),  
    'view' => 'XCalFormatView',  
  );  
  $formatters['ical'] = array(  
    'model' => 'ResourceTimeFeedModel',  
    'mime types' => array('text/calendar'),  
    'view' => 'XCalFormatView',  
    'view arguments' => array('transform'=>'ical'),  
  );  
}
```

# Resursen deklarerar stöd för modellen, inte formatet

```
'models' => array(  
  'ResourceFeedModel' => array(  
    'class' => 'NodeResourceFeedModel',  
  ),  
  'ResourceTimeFeedModel' => array(  
    'class' => 'NodeResourceFeedModel',  
  ),  
)
```

# Flera input-format

- ✦ Inbyggt stöd för x-www-form-urlencoded, yaml, json och serialiserad php.
- ✦ Kan utökas via `hook_rest_server_request_parsers_alter()`.
- ✦ Avgörs av Content-type-headern för anropen och matchas därför via mime-typ:
  - ✦ 'application/json' => 'RESTServer::parseJSON',
  - ✦ 'application/vnd.php.serialized' => 'RESTServer::parsePHP',

# Min syn på services framtid - 3.x

- ♦ De gamla RPC-orienterade metoderna tas bort helt till förmån för resurser.
- ♦ Möjligen stödja översättning av metoddeklarationer till en resurs med actions.
- ♦ Endpoints: moduler, och administratörer, skall kunna publicera och konfigurera servrar på valfri plats i menysystemet.

# Varför endpoints?

Idag publiceras alla installerade tjänster på alla installerade servrar, och alla måste använda samma autentiseringsmodul.

# Varför Endpoints?

- ✦ Detta gör att moduler inte kan använda services för att exponera ett API
- ✦ API = tjänster + server + autentiseringsmetod
- ✦ Eller rättare sagt, endast ett API kan exponeras
- ✦ Detta blir ett problem om services skall användas som ett framework för andra moduler att publicera API:er.

# Endpoints

- ✦ Kan konfigureras oberoende av varandra. Och man kan välja:
  - ✦ vilken server som skall användas
  - ✦ exakt vilka tjänster som skall exponeras
  - ✦ vilken autentiseringsmodul som skall användas och hur den skall vara konfigurerad

# Endpoints gör det möjligt att

- ✦ Exponera flera olika API:er på en Drupal-installation.
- ✦ I moduler definiera ett API som skall bli tillgängligt när modulen installeras.
- ✦ Paketera API:er som features, chaos tools-integration planerat.

# Exempel på endpoint- deklaration

```
/**
 * Implementation of hook_services_endpoints().
 */
function conglomerate_services_endpoints() {
  return array(
    'conglomerate' => array(
      'title' => 'Conglomerate API',
      'server' => 'rest_server',
      'path' => 'api',
      'authentication' => 'services_oauth',
      'authentication_settings' => array(
        'oauth_context' => 'conglomerate',
      ),
      'resources' => array(
        'conglomerate-content' => array(
          'alias' => 'content',
          'operations' => array(
            'create' => array(
              'enabled' => TRUE,
              'oauth_credentials' => 'token',
              'oauth_authorization' => '*',
            ),
            'retrieve' => array(
              'enabled' => TRUE,
              'oauth_credentials' => 'unsigned_consumer',
              'oauth_authorization' => 'read',
            ),
          ),
        ),
      ),
    ),
  ),
);
```

Good=Old

# OAuth och Endpoints

- ✦ OAuth har nu stöd för olika kontext.
- ✦ Consumers placeras alltid i ett kontext
- ✦ Autentiseringar blir därför bara giltiga inom detta kontext
- ✦ Varje kontext kan ha egna auktoriseringsnivåer.
- ✦ Endpoints i services kan antingen använda sig av separata OAuth-kontext eller dela kontext

# Deklarera OAuth kontext i kod

```
/**
 * Implementation of hook_oauth_default_contexts().
 */
function conglomerate_oauth_default_contexts() {
  return array(
    'conglomerate' => array(
      '*' => array(
        'title' => 'Yes, I want to connect !appname to !sitename',
        'description' => 'This will allow your site !appname to push content to !sitename',
        'weight' => -1,
      ),
      'read' => array(
        'title' => 'I want to connect, but just to get stuff from !sitename',
        'description' => 'This will allow !appname to fetch content from !sitename, but it will not
allow any information to be pushed to !sitename.',
        'weight' => 0,
      ),
    )
  );
}
```

# Hugo Wetterberg

@hugowett

hugo@goodold.se

<http://github.com/hugowetterberg>