schülerVZ studiVZ meinVZ

Best Practices API's

Max Horváth, Steffen Irrgang, Andre Zayarni

international PHP2009 conference

webtech CONFERENCE 2009

schülerVZ studiVZ meinVZ

Agenda / Was ist zu beachten?

- Grundlagen / Vorüberlegungen
- Request und Response Format
- Authentifizierung und Security
- Performance
- API-Tests und Support

schülerVZ studiVZ meinVZ

VZ Netzwerke und APIs?

- OAuth API für Externe (T-Mobile's My Phonebook, iPhoto Plugin, ...)
- API's für mobile Clients (iPhone, Android, ...)
- OpenSocial REST API

1. Entwurf: RPC-like

- Eigenes Format
- /getUserData?user_id=1
- GET zum Lesen verwenden
- POST zum Schreiben verwenden

Grundlagen - RESTful

- Architekturstil

- Adressierbarkeit, Zustandslosigkeit
- Operationen (PUT, GET, POST, DELETE)
- GET: /User/1
- POST: /Friends/42/23

schülerVZ studiVZ meinVZ

Restful in PHP I

- Keine vollständiger RESTful Support in PHP
- Browser unterstützen nur GET/POST
- PUT nur durch Umwegen abbildbar
  - Daten aus php://input
- X-HTTP-METHOD-OVERRIDE auswerten

- PHP Frameworks: Symfony, Zend

schülerVZ studiVZ meinVZ

Restful in PHP Beispiel

```
if (isset($_SERVER['X_HTTP_METHOD_OVERRIDE'])) {
      $this->requestMethod = $_SERVER['X_HTTP_METHOD_OVERRIDE'];
} else {
      $this->requestMethod = $_SERVER['REQUEST_METHOD'];
}

switch ($_SERVER['REQUEST_METHOD']) {
      case 'GET':
            $this->input = $_GET; break;
      case 'POST':
            $this->input = $_POST; break;
      case 'PUT':
            // read input from php://input and parse parameters
            $this->input = $this->request->getStreamInput(); break;
      case 'DELETE':
            // no input is allowed
            break;
      default:
            throw new Api_Exception('Request method not allowed',  405);
}
```

Restful in PHP Beispiel

```
if (isset($_SERVER['X_HTTP_METHOD_OVERRIDE'])) {
	$this->requestMethod = $_SERVER['X_HTTP_METHOD_OVERRIDE'];
} else {
	$this->requestMethod = $_SERVER['REQUEST_METHOD'];
}
```

schülerVZ studiVZ meinVZ

Restful in PHP Beispiel

```
switch ($_SERVER['REQUEST_METHOD']) {
	case 'GET':
		$this->input = $_GET; break;
	case 'POST':
		$this->input = $_POST; break;
	case 'PUT':
		// read input from php://input and parse parameters
		$this->input = $this->request->getStreamInput(); break;
	case 'DELETE':
		// no input is allowed
		break;
	default:
		throw new Api_Exception('Request method not allowed',  405);
}
```

Restful in PHP Beispiel

```
case 'GET':
     $this->input = $_GET; break;
case 'POST':
     $this->input = $_POST; break;
```

Restful in PHP Beispiel

```
case 'PUT':
     // read input from php://input and parse parameters
     $this->input = $this->request->getStreamInput(); break;
```

Restful in PHP Beispiel

```
case 'DELETE':
     // no input is allowed
     break;
```

Restful in PHP Beispiel

```
default:
     throw new Api_Exception('Request method not allowed',  405);
```

Restful in PHP Beispiel

```
if (isset($_SERVER['X_HTTP_METHOD_OVERRIDE'])) {
      $this->requestMethod = $_SERVER['X_HTTP_METHOD_OVERRIDE'];
} else {
      $this->requestMethod = $_SERVER['REQUEST_METHOD'];
}

switch ($_SERVER['REQUEST_METHOD']) {
      case 'GET':
            $this->input = $_GET; break;
      case 'POST':
            $this->input = $_POST; break;
      case 'PUT':
            // read input from php://input and parse parameters
            $this->input = $this->request->getStreamInput(); break;
      case 'DELETE':
            // no input is allowed
            break;
      default:
            throw new Api_Exception('Request method not allowed',  405);
}
```

Welche Funktionen?

- Notwendiger Subset der Funktionalität der Webapplikation
- Client definiert Anforderung

- Atomare Funktionen
  - GET: /Friends/2 -> Liste der Id‘s der Freunde
  - GET: /User/3 -> Userdaten (Name, Vorname...)

Welche Funktionen?

- Aber:
  - Kompromis zwischen Requestanzahl und Redundanter Daten
- Auf der einen Seite Requestanzahl minimieren

- Auf der anderen Seite API einfach und klar strukturiert halten

- Beispiel: Mobile Devices, Serverbelastung

Request-Format I

- Vorher Überlegen wir die URIs aussehen
- Spätere Änderungen fast unmöglich
- Extra subdomain „api.domain.de“
- Loadbalancing

Request-Format II

- Versionierung (in der URL enthalten) Beispiel: „/V1/“
- URL in Module / Funktionseinheiten unterteilen
- GET: /User/Profile PUT: /Messages/Inbox

Request-Parameter

- Selbstsprechende Parameternamen
  - under_score ODER camelCase
  - gleiche Namen für Parameter verwenden
- Immer user_id, nicht owner_id oder people_id
- Metaparamter von Methodenparametern unterscheiden

Response - Format

- JSON (default)
  - leichtgewichtig
- XML
  - komplexe Strukturen abbildbar
  - Overhead
- Serialisierte Daten, Key Value Paare...

Response - Format

- Verschiedene Varianten anbieten
- HTTP ACCEPT Header
  - application/json, application/xml
- Als Parameter oder Teil der Ressource
  - &format=json
  - /api/xml/friends

Response - Struktur

- Formatstruktur einmal festlegen

- Struktur- Beschreibung mitschicken
- Parser anbieten
- Rückgaben dokumentieren

- Unterscheidung zwischen Meta- und Payload Daten

schülerVZ studiVZ meinVZ

Response - Struktur

- Metadaten sind Protokollinformationen, die allgemeine Informationen (API-Version) und beispielsweise Iterationsattribute enthalten
  - api_version, count, ...
- Payloaddaten beinhalten die eigentlichen Rückgabewerte der API Methode
  - user_id, user_name, ...

schülerVZ studiVZ meinVZ

Beispiel: JSON Format

JSON:

```
{"result":
{
  "meta"     :
      {
          "version"    : <api_version:number>,
          "nonce"      : <request_nonce:string>,
          "success"    : <success_flag:boolean>,
          "count"      : <content_rows_number:number>
          "total"      : <content_rows_total_number:number>
      },
  "content" : <content_data:string|number|boolean|array>,}}
```

XML:

```
<result>
 <meta>
     <version>...</version>
     ...
 </meta>
 <content>...</content>
</result>
```

Response - Statuscodes

- Durch Restful-Architektur bereits definiert
- Werden im Header angegeben

- Statuscodes
  - 200 bei GET
  - 201 bei Created etc.

Response - Fehlercodes

- 400 Bad Request
  - invalide Parameter
- 503 Service Unavailable
  - API ist offline
- 501 Method not implemented
  - Request Method wird nicht unterstützt

Response - Fehlerbeschreibung

- Fehlermeldung und Fehlercode in Response
- Zusätzlich zum HTTP Status Code
- Wichtig für genaue Fehlerbehandlung

"success“ : false,
"content" : {"message" : "Invalid Parameters", "code" : 400.01}

Response - Fehlerbeschreibung

- Detaillierte Fehlerbeschreibung mit Hilfe von SubIndexes
- 401.01 Invalid OAuth Signature
- 401.03 Invalid OAuth Nonce
- 401.06 Access Token Timed Out

Response - Fehlerausgabe

```
set_exception_handler(array($responseObject, 'errorHandler'));

throw new Api_Exception('Invalid OAuth Signature', 401);
```

schülerVZ studiVZ meinVZ

Response - Errorhandler

```
public function errorHandler(Exception $e) {
    if ($e instanceof Api_Exception) {
        $response['message'] = $e->getMessage();
        $response['code'] = $e->getCode();
    } else {
        $response['message'] = 'Internal Server Error';
        $response['code'] = 500;
    }
 header($_SERVER['SERVER_PROTOCOL'] . $response
['code'] . ' ' .  $response['message']);
$this->output($response);
}
```

Authentifizierung I

- Consumer Authentifizierung
- Vergabe von Consumer API Zugangsdaten
- Zugriff auf geschützte Ressourcen
- Endnutzer muss sich authentifizieren

Authentifizierung II

- Varianten:
  - OAuth
  - Sessionbasiert (Session Cookie)
  - HTTP Basic Auth, HTTP Digest

- Plugable
- Model muss unabhängig von der Authentifizierung sein (MVC Pattern)

schülerVZ studiVZ meinVZ

OAuth

- Offenes Protokoll
- Zugriff auf geschützte Ressourcen

Rollen

- Client - OAuth Consumer
- API Server - OAuth Provider
- User

- API Methoden - geschützte Resourcen

OAuth

- Autorisierung anhand vom authentifizierten Token

- Consumer fordert Request Token an
- User authentifiziert Token auf Provider Seite
- Consumer tauscht Request Token gegen Access Token ein
- API Zugriffe mit Access Token

OAuth

- Weit verbreitet
  - Google, Yahoo, Twitter ...
- Kein SSL notwendig
  - Requests werden mit OAuth Signatur signiert (HMAC_SHA1, RSA-SHA1)
- Endbenutzer Login Daten werden nicht dem Consumer preisgegeben

OAuth

- Erfordet User Interaktion im Browser
  - Zweibeiniges OAuth
- Phishing
- PHP Implementierungen: Google Code Projekt, Zend (Incubator), PHP 5.3

schülerVZ studiVZ meinVZ

Privacy

- Trennung Viewer und Owner
- Privacy Layer

schülerVZ studiVZ meinVZ

Privacy

- Trennung Viewer und Owner
- Privacy Layer

- Deutsches Datenschutzgesetz
  - Schaf-schick-Problem
  - User muss dem Dienst erst zustimmen

schülerVZ studiVZ meinVZ

Privacy

- Trennung Viewer und Owner
- Privacy Layer

- Deutsches Datenschutzgesetz
  - Schaf-schick-Problem
  - User muss dem Dienst erst zustimmen

- VZ Lösung: Nutzer zu Diensten einladen
- Nachteil: hohe Barriere für den Dienst

schülerVZ studiVZ meinVZ

Security

- vorhandene Webfeatures nicht nutzbar
  - Captchas sind nicht möglich

schülerVZ studiVZ meinVZ

Security

- vorhandene Webfeatures nicht nutzbar
  - Captchas sind nicht möglich

- Ratelimits einbauen
  - pro Consumer und pro User

schülerVZ studiVZ meinVZ

Security

- vorhandene Webfeatures nicht nutzbar
  - Captchas sind nicht möglich
- Ratelimits einbauen
  - pro Consumer und pro User
- Quota (Begrenzung Trafficvolumen)
- Repeatattacken verhindern (durch Nonce)

API Tests

- Communication Tests durchführen
- Keine Businesslogik testen

schülerVZ studiVZ meinVZ

API Tests

- Communication Tests durchführen
- Keine Businesslogik testen

```
public function testGetPhotoAlbums() {
    $this->_doLogin();
    $this->_http->setProperty('path', $path . 'UserAlbums/' . $id);
    $response = $this->_getRequest();
    $this->_assertContains($response, '"success":true');
}
```

Performance

- Loadbalancing (dedizierte API Server)

Performance

- Loadbalancing (dedizierte API Server)

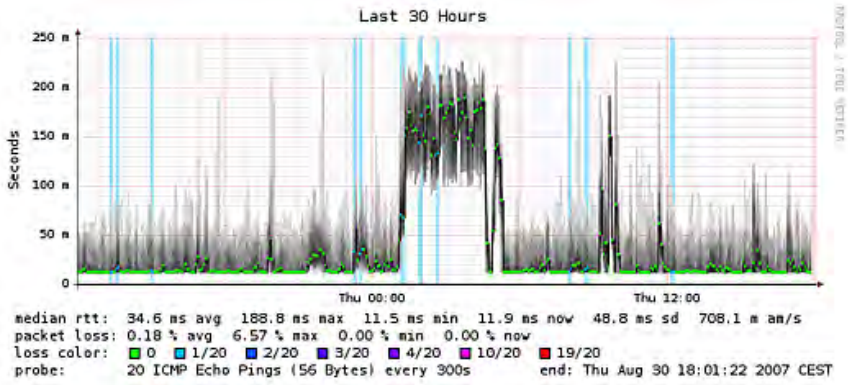
- Caching
  - Framework internes Caching
  - Response Caching (Zeit- und Requestabhängig)
  - Clientseitiges Caching empfehlen

schülerVZ studiVZ meinVZ

Monitoring

- Fehlerlogs

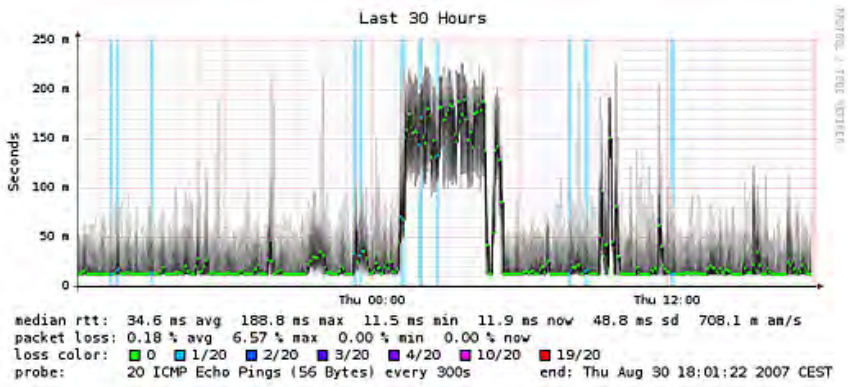
- Vergabe und Verwaltung von Consumer
  - Tracken von Requests



schülerVZ studiVZ meinVZ

Monitoring

- Fehlerlogs
- Vergabe und Verwaltung von Consumer
  - Tracken von Requests
- Latenzzeiten
  - Nagios, Smokeping



schülerVZ studiVZ meinVZ

Developer Support

- Dokumentation, HowTo's und Wiki

schülerVZ studiVZ meinVZ

Developer Support

- Dokumentation, HowTo's und Wiki
- Bereitstellung von SDK
- Beispielclients für Frameworks

schülerVZ studiVZ meinVZ

Developer Support

- Dokumentation, HowTo's und Wiki
- Bereitstellung von SDK
- Beispielclients für Frameworks
- Developercommunity aufbauen
  - Entwickler helfen sich selbst
  - Iterative Verbesserung der API

schülerVZ studiVZ meinVZ

Fragen?

Danke fürs Zuhören!

Kontakt: developer.studivz.net

studivz.net/max | twitter.com/MaxHorvath

studivz.net/steffen | twitter.com/sirrgang

studivz.net/andre | twitter.com/andre_z