AJAX IN ACTION

Offline-Webanwendungen mit Google Gears

Schuster Stefan - Irian.at

Stefan Schuster

- Web 2.0 / Ajax Entwickler

irian | THE JAVA EXPERTS

mind42.com BETA

Agenda

- Theorie
  - Google Gears
  - Dojo Offline
  - Applikations-Architektur
- Praxis
  - Ein Beispiel: MultiUserTodoTable

AJAX IN ACTION

Theorie: Google Gears

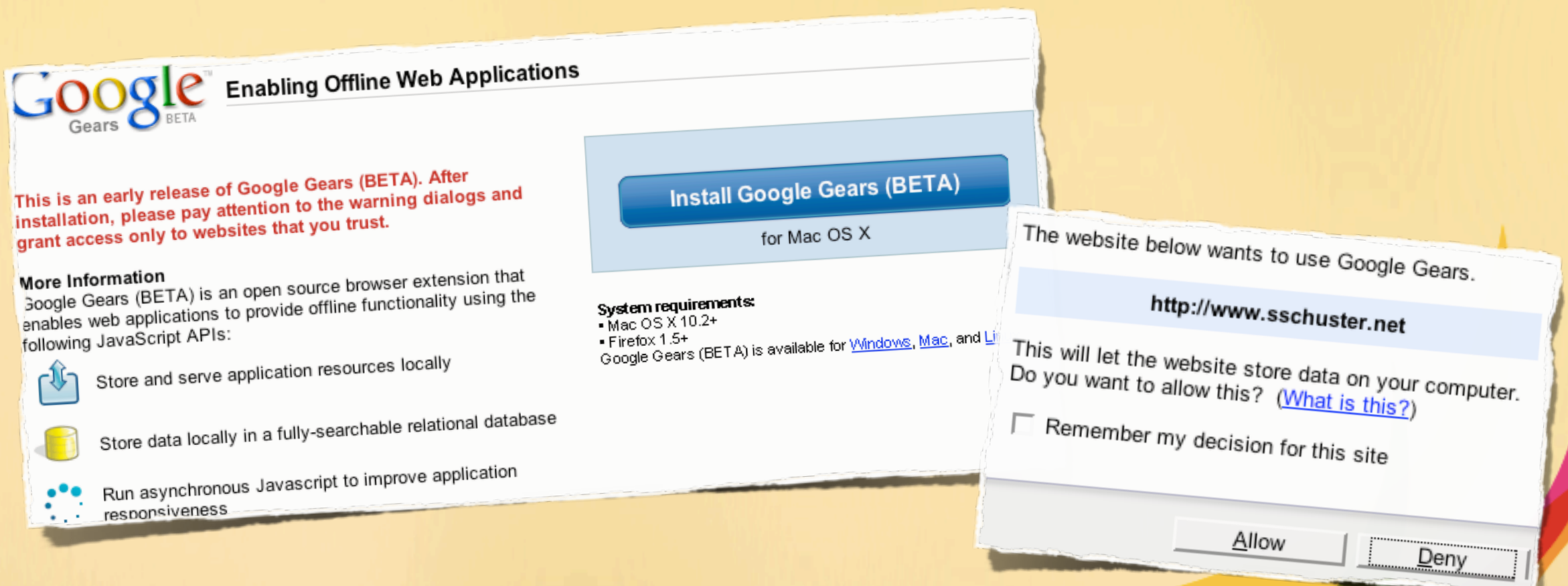
Google Gears

- Browserplugin

- Ermöglicht Offline-Webapplikationen
- Erschien im Rahmen des Google Developer Day im Mai 2007

Google Gears

- Webseiten müssen Google Gears unterstützen
- Gears muss “erlaubt” werden



Google Gears

- 3 Komponenten
  - LocalServer 
  - Database 
  - WorkerPool 

LocalServer

- Caching der Applikation
  - HTML
  - JS
  - CSS
  - Bilder

LocalServer

- ResourceStore
  - Manuelles Caching & Updating mittels JS-API
  - FileSubmitter
    - “Offline” hochladen von Dateien
- ManagedResourceStore
  - Automatisches Caching & Updating mittels Manifest-File

LocalServer

- Unabhängig vom On/Offline Status
- LocalServer liefert URLs aus dem Cache wenn
  - Im ResourceStore angegeben
  - ResourceStore enabled

Database

- Relationale Datenbank
  - SQLite
- JS-API
  - Bind-Parameters
- Erweitert um Volltextsuche (fts2)
- Same-Origin Security-Policy
  - Nur Zugriff auf Datenbanken einer “Domain”
    - ATTACH & DETACH deaktiviert

WorkerPool

- Separate JS-Prozesse
  - Hauptprozess im Browser blockiert UI bei langen Berechnungen
  - In “worker” ausgegliederter Code läuft parallel & unabhängig
    - blockiert nicht
  - Kein DOM
  - Kein XMLHttpRequest

AJAX IN ACTION

Theorie: Dojo Offline

Dojo Offline

- Ursprünglich proprietäres System mit eigenem Proxy

dōjō offline
small. easy. open.

- Mittlerweile aufbauend auf Google Gears
- Higher-Level API für Gears
- Teil von dojox (experimentelle Dojo 0.9+ Modulsammlung)

Dojo Offline

- Kapselt und erweitert Google Gears
- Bestandteile
  - Framework
  - Synchronisation / Actionlog
  - Status Widget
  - dojox.sql

Framework

- Automatisches Einleiten der Vorgänge
  - Überprüfung Online/Offline
  - Start Datendownload
  - Start Synchronisation
- Automatische Pflege des ResourceStore
  - slurp()

Synchronisation

- Guidelines:
  - Automatische Synchronisation (keine Benutzerinteraktion)
  - Keine Merge-Interfaces (intelligente Entscheidungen treffen)
  - User nur über besondere Vorfälle informieren

Actionlog

- Jede Offline Aktion wird als Action persistiert
  - Command Pattern
  - Alle notwendigen Daten um die Aktion zu wiederholen müssen gespeichert werden
- Beim Upload werden die Actions in der richtigen Reihenfolge abgespielt

Synchronisation

- Aktionen
  - slurp()
  - Upload / Actionlog
  - Download

- Ablauf von Dojo Offline gesteuert
  - Anstoß durch Events
  - Callbacks für den Ablauf

Status Widget

- Philosophie: Online/Offline Mechanismen transparent gestalten
- User lediglich über das wichtigste informieren
  - Installation
  - Online
  - Offline
  - Synchronisation

dojox.sql

- Öffnet/Schließt Datenbank automatisch
- Verschlüsseln von Daten
  - ENCRYPT/DECRYPT Keywords
  - WorkerPool
- Alternativ: dojox.storage
  - key/value Hash

AJAX IN ACTION

Theorie: Applikations-Architektur

Applikations-Architektur

- Was haben wir?
  - Cache für Daten - Gears
  - Automatisches Caching - Dojo
  - Relationale Datenbank - Gears/Dojo
  - Key/Value Datenspeicher - Dojo
  - Mehrere Prozesse - Gears
  - Statusverwaltung - Dojo
  - Synchronisations-Framework - Dojo

Applikations-Architektur

- Was brauchen wir zur lauffähigen Applikation?
  - Online/Offline Datenzugriff?
  - Offline View-Generierung?
  - Synchronisation - Wo verarbeiten wir die Dojo Actions?

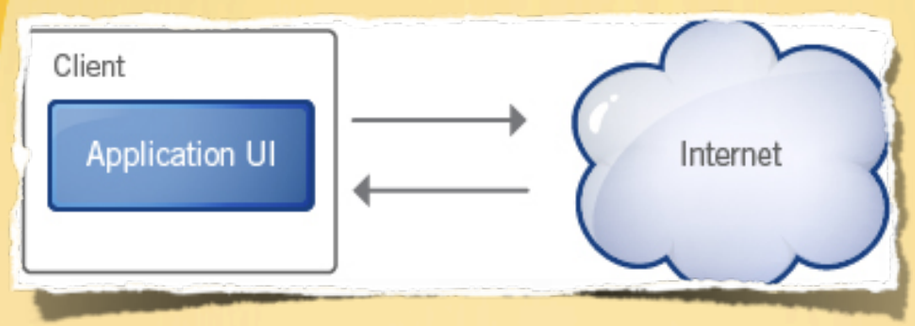
Datenzugriff

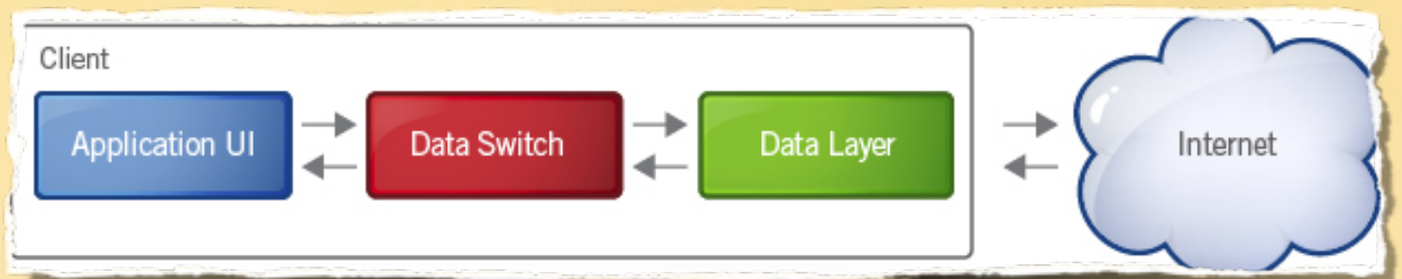
- Szenarien
  - Browser ohne Gears-Unterstützung
  - Browser mit Gears Online
  - Browser mit Gears Offline
- Online/Offline Varianten
  - Transparent (Modeless)
  - Expliziter Statuswechsel (Modal)

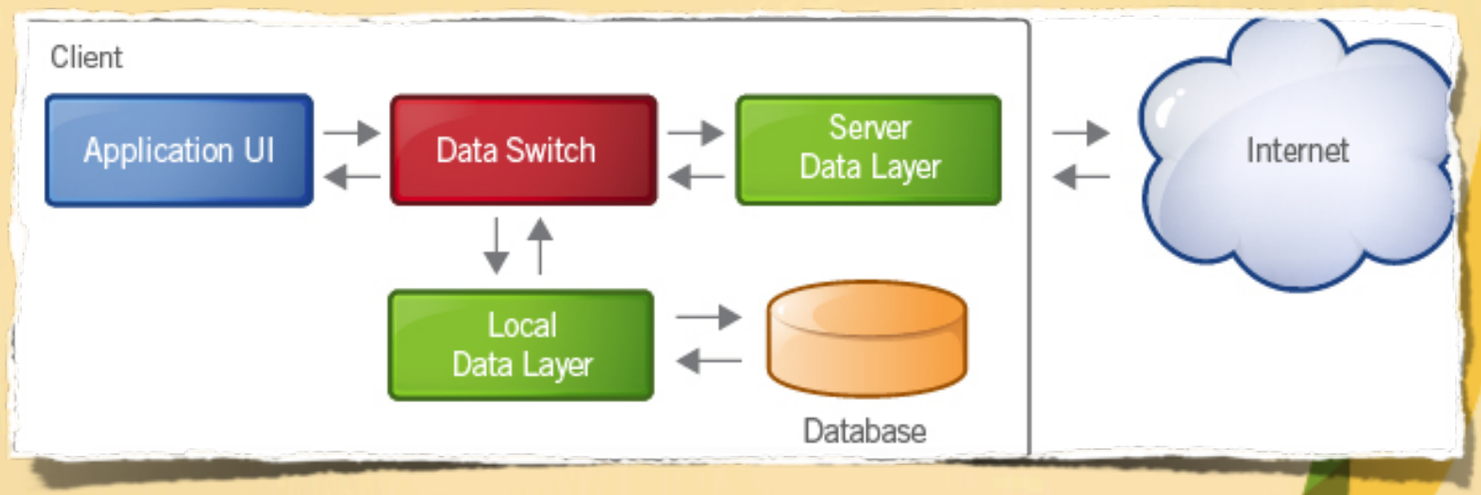
Datenzugriff

- Google gibt nur theoretische Anregungen
- Dojo stellt keinerlei Unterstützung zur Verfügung

Datenzugriff

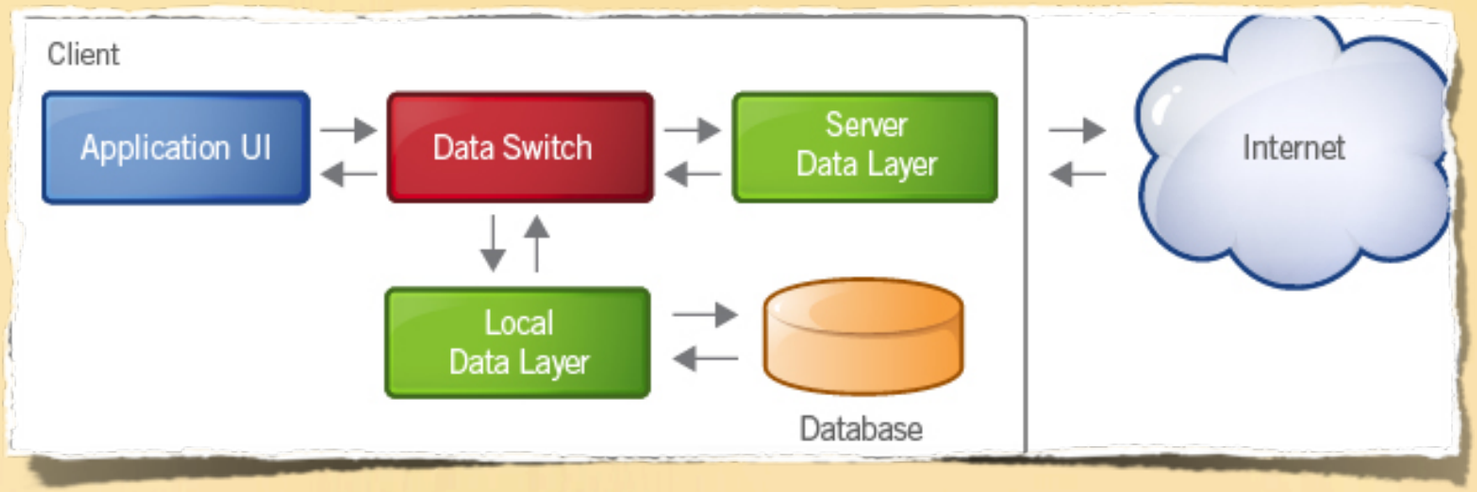






Datenzugriff

- Beste Variante:
  - Modeless für den User
  - Modeless für die Applikation



Offline-Views

- Je nach Applikation/Features
  - Nicht alle Funktionen Offline umsetzbar
- Voraussetzung für Offline-Funktionalität
  - Views müssen Offline generiert werden
  - Keine Server-berechneten HTML Snippets

Synchronisation

- Eines der komplexesten IT-Themen überhaupt
- Aufbauend auf dojox.offline Actionlog
  - Wie müssen solche Actions aussehen?
  - Wie verarbeiten wir Actions am besten
    - Ziel != monolithische Sync.-Methode
  - Wie gehen wir mit IDs um?

AJAX IN ACTION

Praxis: Beispiel MUTT

MultiUserTodoTable

MUTT

- MultiUserTodoTable
  - Einfache Todo-Liste / Issue-Tracker
- Zeigt
  - Data-Switch
  - Offline View-Generierung
  - Einfache Synchronisation

AJAX IN ACTION

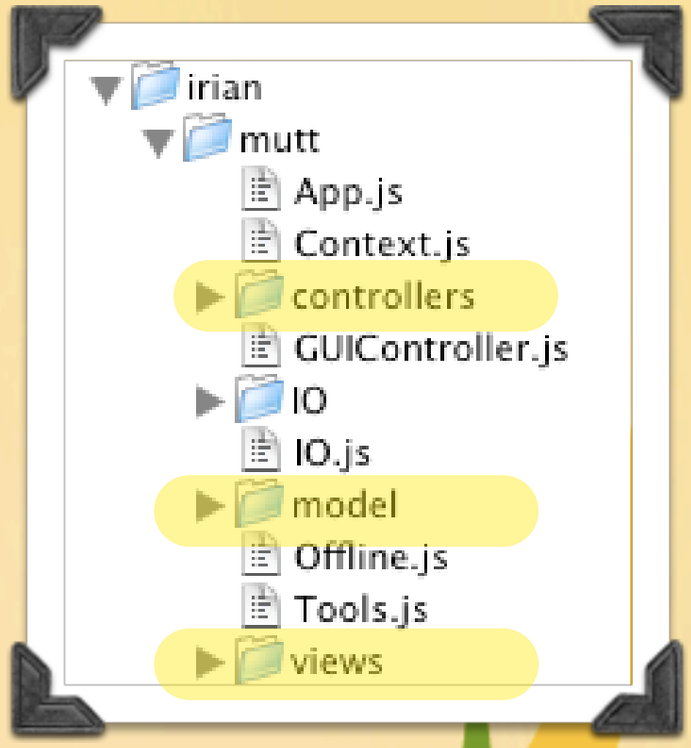
DEMO

MUTT

- Code-Walkthrough
  - MVC Architektur
  - Offline View-Generierung
  - Dataswitch - Verwendung
  - Dataswitch - Implementierung
  - Synchronisation

MVC Architektur

- Models
  - Kapselung aller Daten in Model-Objekte
  - Dataswitch auf Model-Ebene
- Views
  - Clientside View Generierung
- Controller
  - Event Handling der Views



Model

```
dojo.declare(
        "irian.mutt.model.User",
        null,
        {
                constructor: function(id, name) {
                        this.id = id || Math.NaN;
                        this.name = name || "No name";
                },
                getId: function() {
                        return this.id;
                },
                setId: function(id) {
                        this.id = id;
                },
                getName: function() {
                        return this.name;
                },
                setName: function(name) {
                        this.name = name;
                }
        }
);
```

Offline View-Generierung

- Views werden komplett am Client gerendert
- Google Trimpath Javascript Templates

```
this.div.childNodes[3].innerHTML = [
    '{if comments.length > 0}',
    '<h3>Comments</h3>',
    '{for comment in comments}',
       '<div class="comment">',
       '<div class="username">${comment.getCreator().getName()}</div>',
       '<div class="text">${comment.getText()|escape|nl2br}</div>',
       '</div>',
    '{/for}',
    '{/if}'
].join("").process({
    comments: comments,
    _MODIFIERS: irian.mutt.Tools.jstCommentModifiers
});
```

Dataswitch - Verwendung

- Um Daten in einem Controller abzurufen

```
mutt.IO.access.todo.get(todoId, dojo.hitch(this, function(todo) {
    this._show(parentNode, todo);
}));
```

- mutt.IO.access beinhaltet je nach Status automatisch die richtigen Access-Objekte
- Rückgabewert als Model-Objekt im Callback

Dataswitch - Implementierung

- mutt.IO.access ist eine Referenz auf eines von 3 Objekten
  - accessOnline
  - accessOnlineCache
  - accessOffline

- Dojo Events werden verwendet um mutt.IO.access auf das richtige zeigen zu lassen

Dataswitch - Implementierung

```
constructor: function(server) {
   this.accessOnline = {
      user: new irian.mutt.IO.UserOnline(),
      todo: new irian.mutt.IO.TodoOnline(),
      comment: new irian.mutt.IO.CommentOnline()
   };
   this.accessOnlineCache = {/*****/};
   this.accessOffline = {/*****/};
   this.access = this.accessOnline;
   this.server = server;
},
changeState: function() {
   if (dojox.off.enabled) {
      if (dojox.off.isOnline) {
         this.access = this.accessOnlineCache;
      }
      else {
         this.access = this.accessOffline;
      }
   } else {
      this.access = this.accessOnline;
   }
}
```

Dataswitch - Implementierung

- Beispiel: UserOnline.get()

```
get: function(id, callback) {
   /* CACHING */
   var that = this;
   dojo.xhrPost({
      url: mutt.IO.server + "user/get.php",
      content: {
         id: id
      },
      handleAs: "json",
      handle: function(response) {
         /* ERROR HANDLING & CACHING */
         var user = new irian.mutt.model.User(response.id, response.name);
         callback(user);
      }
   });
}
```

Dataswitch - Implementierung

- Beispiel: UserOnlineCache.get()
  - Überschreibt UserOnline.get()

```
get: function(id, callback) {
   var that = this;
   irian.mutt.IO.UserOnline.prototype.get.call(this, id, function(user) {
      dojox.sql(
         "INSERT OR IGNORE INTO users VALUES (?,?)",
         user.getId(), user.getName());
      callback(user);
   });
}
```

Dataswitch - Implementierung

- Beispiel: UserOffline.get()

```
get: function(id, callback) {
    /* CACHING */
    var result = dojox.sql("SELECT name FROM users WHERE id=?", id)[0];
    /* ERROR HANDLING
    var user = new irian.mutt.model.User(id, result.name);
    callback(this.userCache[id]);
}
```

Synchronisierung

- Part 1: Action Erstellung
- Part 2: Action Replay

Action Erstellung

- Beispiel: UserOffline.create()

```
create: function(name, callback) {
   //Create user in DB
   var result = dojox.sql("SELECT MIN(id) AS id FROM users")[0];
   var id = -1;
   if (result) {
      id = Math.min(-1, --result.id);
   }
   dojox.sql("INSERT INTO users VALUES(?, ?)", id, name);

   //Create sync action
   dojox.off.sync.actions.add({
      module: "user",
      method: "syncCreate",
      data: {
         offlineId: id,
         name: name
      }
   });
   //...
```

Action Replay

- Alle Actions in MUTT haben den Aufbau
  - module
  - method
  - data
- Offline erstellte Datensätze haben negative IDs
- Ermöglicht generische Actionlog Replay Methode

Action Replay

```
onActionReplay: function(action, actionLog) {
   mutt.IO.accessOffline[action.module][action.method](
      action.data,
      function(success, msg) {
         if (!success && dojo.isString(msg)) {
            actionLog.haltReplay(msg);
         }
         else {
            actionLog.continueReplay();
         }
      });
   }
}
```

Action Replay

- Sync-Methoden verwenden OnlineCache um alle Actions zu wiederholen
  - Erweitert um Error Handling
  - ID Mapping
    - Lokal: Negative ID
    - Server: Positive ID
    - Für Updates & Relationen muss während der Synchronisation ein Mapping vorhanden sein

Action Replay

```
syncCreate: function(data, callback) {
   /* USER ALREADY EXISTS CHECK ... */

   var that = this;
   mutt.IO.accessOnlineCache.user.create(data.name, function(user) {
      if (!user) {
         callback(false, "Error creating user '" + data.name + "'");
      }
      else {
         that.offlineIdMap[oldId] = newUser.getId();
         dojox.sql("DELETE FROM users WHERE id=?", oldId);
         callback();
      }
   });
}
```

MUTT

- Live-Demo & Download

http://www.sschuster.net/mutt

AJAX IN ACTION

Vielen Dank für Ihre Aufmerksamkeit