(Google) Gears

Seminarvortrag: Marina Eins

Inhalt

- Gears Allgemein
  - Motivation, Definition, Hintergrund, Voraussetzungen
- Gears auf Mobilgeräten
- Gears-Komponenten
- Gears-Funktionsweise
- Eine Web-Anwendung Gears-fähig machen: Voraussetzungen
- Eine Web-Anwendung Gears-fähig machen: Die 4 Schritte
- Eine Web-Anwendung Gears-fähig machen: Ein Beispiel
- Sicherheit und Risiko bei Gears
- Ausblick: HTML5 vs. Gears

Motivation

Online-Webapplikationen:

- müssen nicht installiert werden
- benötigen als Voraussetzung nur Webbrowser mit Internetzugang
- neue Features stehen ohne komplizierte Updates zur Verfügung

Offline-Webapplikationen?

- bei langsamer oder instabiler Internetverbindung
- bei fehlender Internetverbindung

Definition

(Google) Gears: **(dt. „Getriebe“)**

- Open-Source-Software und
- Browser-Plugin von Google Inc

- Erweitert Web-Browser, so dass Online-Webanwendungen auch im Offline-Betrieb genutzt werden können

→ Nutzung der Anwendungen zu jeder Zeit (offline)
→ Verbesserung der Performance

Gears

Hintergrund

Erstmalig vorgestellt auf dem
Google Developer Day 2007 in Sydney

(http://www.youtube.com/watch?v=cQyha30nm6k)

Seit Juni 2008 statt „Google Gears“ nur noch „Gears“

Voraussetzungen

für die Benutzung

- Gears-Plugin für den Browser (http://gears.google.com)

  Firefox 1.5+ für Windows, Mac OS X, Linux
  Internet Explorer 6.0+ für Windows
  Internet Explorer Mobile 4.01+ für Windows Mobile 5+
  Safari 3.1.1+ für Mac OS X
  Android-Browser auf Android-Geräten haben Gears schon integriert

- eine Gears-kompatible Website bzw. Anwendung

  Google Mail (E-Mail-Dienst)
  Google Kalender (Webkalender)
  Remember the milk (Aufgabenverwaltung)
  Youtube (Videoupload)
  Studivz (Fotoupload)
  …

Gears auf Mobilgeräten

- Verfügbar für

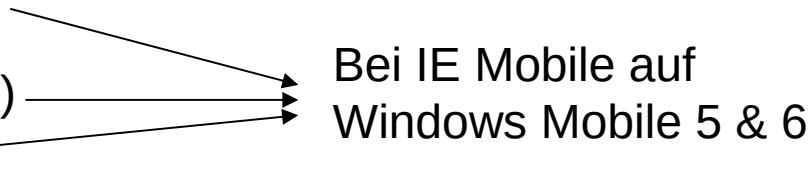
  Windows Mobile 5 & 6 und Android

- Gleiche API wie Desktop-Version

- Gleiche Funktionalität wie Desktop-Version

- Einschränkungen nur durch das Gerät/OS an sich:

  - kleinerer Bildschirm, limitierte Texteingabe
  - Cascading Style Sheets (CSS)
  - Document Object Model (DOM)
  - ActiveX



Genaueres unter: http://code.google.com/intl/de-DE/apis/gears/mobile.html

Komponenten

Gears besteht aus zwei Komponenten:

- Browser-Plugin: ermöglicht den APIs den Zugriff auf den lokalen Datenträger

- API-Sammlung: lässt sich per Javascript ansprechen, ist in 9 Module unterteilt

  - **Factory**: dient zum Instantiieren aller anderen Gears-Objekte.
  - **LocalServer:** dient dem lokalen Speichern von Anwendungs-Ressourcen
  - **Database:** dient lokalem Speichern von Anwendungs-Daten in einer Datenbank
  - **WorkerPool:** dient dem *asynchronen* (im Hintergrund) ausführen von Javascript
  - **Geolocation:** dient der Standortbestimmung des Clients
  - **Desktop:** dient dem Anlegen von Verknüpfungen auf dem Desktop
  - **HttpRequest:** ermöglicht HTTP-Anfragen
  - **Timer**: kann Zeitschalter setzen sowie auf sie reagieren
  - **Blob**: ist eine Javascript-Klasse für Binärdaten

Übersicht aller APIs und des Source-Codes unter: **http://code.google.com/p/gears/**

Das Factory-Modul

Die Factory-Klasse dient zum instantiieren aller anderen Gears-Objekte.

Über eine „create“-Methode legt sie ein Objekt der jeweils übergebenen Klasse an.

Beispiel: Anlegen eines Datenbank-Objektes

```
<script type="text/javascript" src="gears_init.js"></script>

<script type="text/javascript">

if (window.google && google.gears)

{ var db = google.gears.factory.create('beta.database');  db.open(); }

</script>
```

„gears_init.js“ definiert die Factory, muß daher in jeder Gears-Anwendung implementiert werden (Gehört zur Gears-Distribution) (Ist auf der Gears-Homepage erhältlich)

Prüft, ob Gears installiert ist

Über „create()“ ein Database-Objekt instantiieren

Das LocalServer-Modul

Die LocalServer-Klasse ermöglicht es einer Webanwendung ihre Anwendungs-Ressourcen (HTML, CSS, JavaScripte, Bilder etc.) in einem lokalen Store zu cachen. Dadurch wird Offline-Funktionalität ermöglicht!

Für das Caching der Anwendungs-Ressourcen gibt es 2 verschiedene Stores:

- **ResourceStore:** für manuelles Caching der Ressourcen über Javascript
- **ManagedResourceStore:** für manuelles & automatisches Caching der Ressourcen über ein Manifest-File.

Manifest-File: Ein TXT-File, das alle URLs der zu cachenden Anwendungs-Ressourcen auflistet.
Die im Manifest gelisteten Files können manuell über die „checkForUpdate()“-Methode der ManagedResourceStore-Klasse und automatisch bei jeder Anfrage nach Ressourcen aus dem ManagedResourceStore gecached bzw. geupdatet werden.

Das LocalServer-Modul

Beispiel: Cachen von Files in einen ManagedResourceStore

```
<script type="text/javascript„ src="gears_init.js"></script>
<script type="text/javascript">
var localServer = google.gears.factory.create('beta.localserver');
var store = localServer.createManagedStore('test-store');
store.manifestUrl = 'site-manifest.txt';
store.checkForUpdate();
</script>
```

„gears_init.js“ einbinden

LocalServer-Objekt anlegen

ManagedResourceStore über den LocalServer anlegen

Dem ManagedResourceStore sein Manifest-File zuweisen

Files aus dem Manifest-File cachen bzw. updaten

```
// Inhalt von „site-manifest.txt“
{ "betaManifestVersion": 1,
"version": "version 1.0",
"entries": [ { "url": "site.html" }, { "url": "gears_init.js" } ] }
```

Version des Manifest-File-Formats (1 oder 2)

String, der die Version der Ressourcen-Sammlung angibt

Die zu cachenden Ressourcen

Das Database-Modul

Die Database-Klasse dient zum lokalen Speichern von Anwendungs-Daten in einer SQLite-Datenbank (relationale SQLfähige DB).

Die Daten werden über SQL-Statements in einer „execute“-Methode gespeichert, abgerufen und geändert.

Beispiel: Speichern und Abrufen von Daten in ein einer Datenbank

```
<script type="text/javascript" src="gears_init.js"></script>

<script type="text/javascript">
var db = google.gears.factory.create('beta.database');
db.open('database-test');
db.execute('create table if not exists Test (Phrase text, Timestamp int)');
db.execute('insert into Test values (?, ?)', ['Monkey!', new Date().getTime()]);
var rs = db.execute('select * from Test order by Timestamp desc');
while (rs.isValidRow()) { alert(rs.field(0) + '@' + rs.field(1)); rs.next(); }
rs.close();
</script>
```

Database anlegen

Legt eine Tabelle „Test“ in der Datenbank an

Befüllt „Test“-Tabelle

Ruft Daten aus der Test-Tabelle ab und gibt das ResultSet jeder Zeile durch „@“ verbunden aus

Das WorkerPool-Modul

Die WorkerPool-Klasse ermöglicht Web-Anwendungen das Ausführen von JavaScript-Code (bzw. Child-Workern) im Hintergrund, ohne dass die Ausführung des Hauptseiten-Script (bzw. Parent-Worker) geblockt wird.
Worker kommunizieren miteinander nur über das Senden von Message-Objekten.

Beispiel: WorkerPool anlegen und Worker miteinander kommunizieren lassen

```
// main.js (bzw. parent-worker)
var workerPool = google.gears.factory.create('beta.workerpool,);
workerPool.onmessage = function(a, b, message) {
alert('Received message from worker ' + message.sender + ': \n' + message.body); };
var childWorkerId = workerPool.createWorkerFromUrl('worker.js');
workerPool.sendMessage(["3..2..", 1, {helloWorld: "Hello world!"}], childWorkerId);

// worker.js (bzw. child-worker)
var wp = google.gears.workerPool;
wp.onmessage = function(a, b, message) {
var reply =
message.body[0] + message.body[1] + "... " + message.body[2].helloWorld;
wp.sendMessage(reply, message.sender); }
```

WorkerPool anlegen

Wenn Message im WorkerPool eingeht folgt Alert

Child-Worker mit JS-Code aus einer URL anlegen

Message an Child-Worker senden

Wenn Message im Child-Worker eingeht, Antwort erzeugen

Antwort an den Sender senden

Das Geolocation-Modul

Die Geolocation-Klasse ermöglicht einer Web-Anwendung die geographische Position eines Nutzers zu bestimmen und zu ermitteln, wenn sich seine Position ändert.

Die Positionsbestimmung erfolgt anhand von GPS, WLAN oder Sendemasten über deren Funkstärke die Position errechnet wird.

Beispiel: Längen- und Breitengrad der aktuellen Position ermitteln

```
var geo = google.gears.factory.create('beta.geolocation');
function updatePosition(position) {
 alert('Current lat/lon is: ' + position.latitude + ',' + position.longitude); }

function handleError(positionError) {
 alert('Attempt to get location failed: ' + positionError.message); }

geo.getCurrentPosition(updatePosition, handleError);
```

Geolocation-Objekt anlegen

Über „updatePosition()“ Längen- und Breitengrad abrufen und ausgeben

Über „handleError()“ Fehlermeldung ausgeben

Über die Geolocation-Methode „getCurrentPosition()“ die Callback-Funktionen für erfolgreiche oder nicht erfolgreiche Positionsbestimmung angeben

Das Desktop-Modul

Die Desktop-Klasse ermöglicht den Zugriff auf Desktop-bezogene Funktionalitäten, wie das Anlegen eines Shortcuts auf dem User-Desktop oder die benutzergesteuerte Auswahl lokaler Files für die Verwendung in einer Web-Anwendung.

Beispiel: Anlegen eines Shortcuts und Auswahl lokaler Files

```
var desktop = google.gears.factory.create('beta.desktop');
desktop.createShortcut(
'Test Application',
'http://example.com/index.html',
{'128x128': 'http://example.com/icon128x128.png', '48x48': 'http://example.com/icon48x48.png',
'32x32': 'http://example.com/icon32x32.png', '16x16': 'http://example.com/icon16x16.png'},
'An application at http://example.com/index.html');

function openFilesCallback(files) { alert('User selected ' + files.length + ' files.'); }

desktop.openFiles(openFilesCallback);
```

Desktop-Interface anlegen

Desktop-Shortcut erzeugen (Übergabe von Name, URL, Icons, Beschreibung)

Callback-Funktion zur Ausgabe der Anzahl der ausgewählten Benutzer-Files

Desktop-Methode zum öffnen eines Auswahl-Dialoges, in dem der User seine lokalen Files auswählen kann, um sie in einer Webanwendung verfügbar zu machen. Nach der Auswahl wird die entsprechende Callback-Funktion aufgerufen

Das HttpRequest-Modul

Die HttpRequest-Klasse ermöglicht das Senden von HTTP Anfragen (z.B. GET, POST, HEAD, PUT, ..), sowohl von der Haupt-HTML-Seite als auch innerhalb von Workern.

Beispiel: GET-Anfrage

```
<script type="text/javascript" src="gears_init.js"></script>
<script type="text/javascript">

var request = google.gears.factory.create('beta.httprequest') ;

request.open('GET', '/index.html');

request.onreadystatechange = function() {

if (request.readyState == 4) { console.write(request.responseText); } };

request.send();

</script>
```

HttpRequest-Interface anlegen

Methode und URL eines Requests spezifizieren

Funktion spezifizieren, die bei Statusanzeige 4 (=Complete) des Requests, einen Antwort-Text auf der Console ausgibt

Den Request senden

Das Timer-Modul

Die Timer-Klasse ermöglicht das Setzen und Reagieren von Zeitschaltern, sowohl für die Haupt-HTML-Seite als auch innerhalb von Workern.

Beispiel: Alert über einen Timer ausgeben

```
<script type="text/javascript" src="gears_init.js"></script>

<script type="text/javascript">

var timer = google.gears.factory.create('beta.timer');

timer.setTimeout(function() { alert('Hello, from the future!'); }, 1000);

</script>
```

Timer anlegen

Setzt Timeout, der nach 1000 Millisekunden eine Funktion aufruft, die eine Message ausgibt

Das Blob-Modul

Die Blob-Klasse ermöglicht das Referenzieren von Blobs bzw. Blöcken von Binärdaten (z.B. Grafikdateien, Audiodateien), in Web-Anwendungen.
Blobs können so über Methoden anderer Gears-APIs verwendet werden.

Beispiel: Blob-File-Upload über HttpRequest

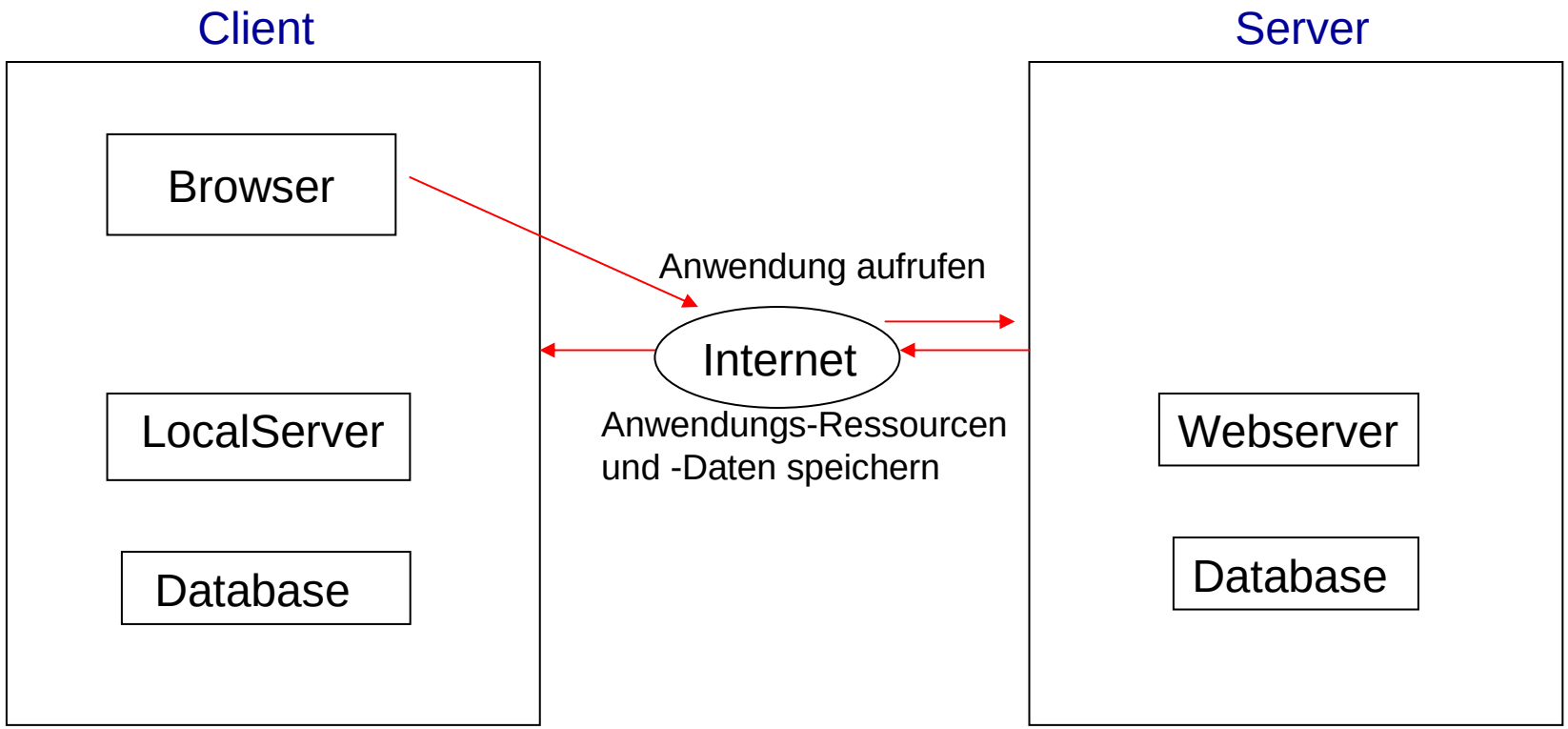
```
<script type="text/javascript" src="gears_init.js"></script>

<script type="text/javascript">

var desktop=google.gears.factory.create('beta.desktop');
desktop.openFiles(function(files){

var http = google.gears.factory.create('beta.httprequest');
http.open('PUT', 'http://server/'+files[0].name);
http.onreadystatechange = function() {
if (http.readyState == 4) { alert(http.responseText); }};

http.send(files[0].blob);
}
</script>
```

Desktop-Interface anlegen

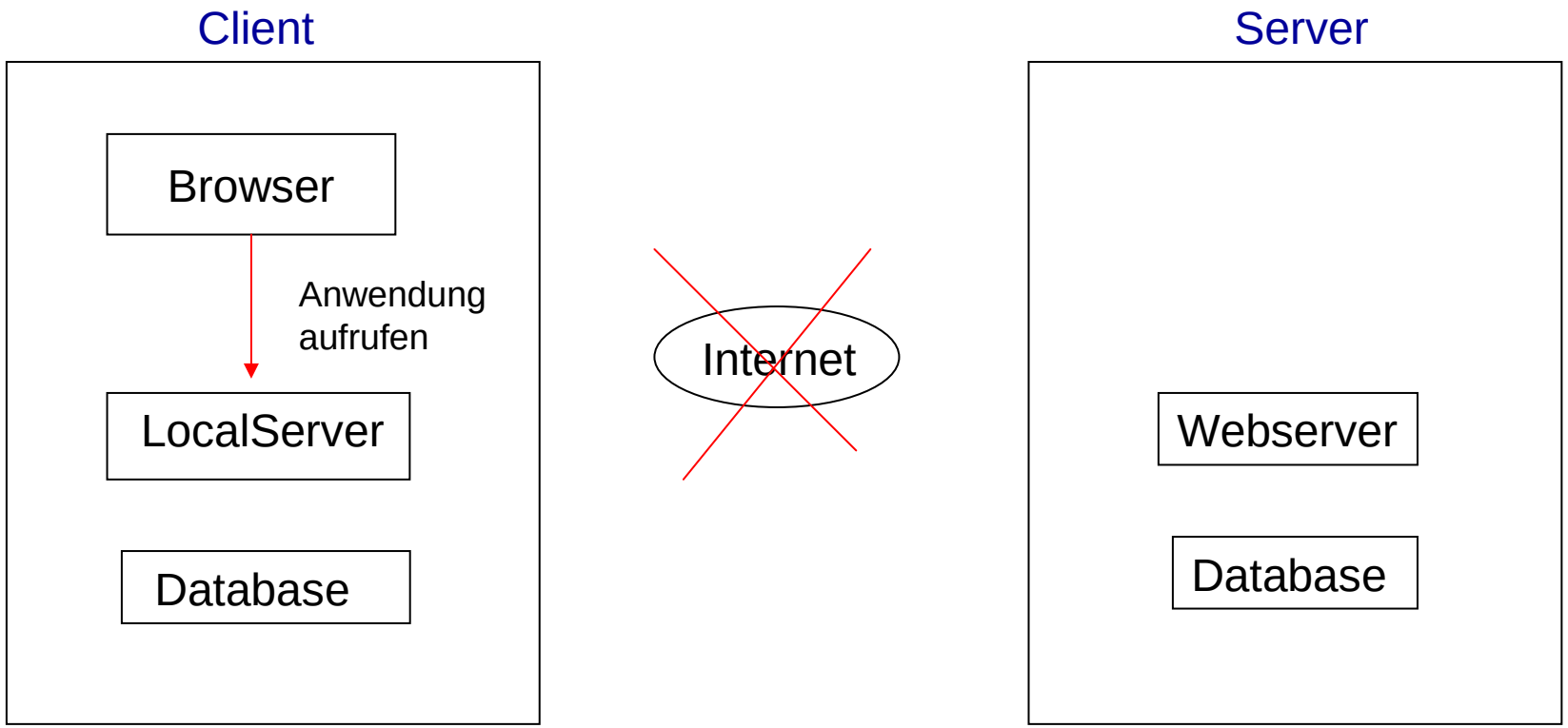
Mit einer Desktop-Callback-Funktion einen HttpRequest zum uploaden eines ausgewählten Files definieren.

Den HttpRequest für das Uploaden des ausgewählten Blob-Files senden

Funktionsweise
online



Funktionsweise
offline



Gears

Voraussetzungen

für die Entwicklung

- Fähigkeit JavaScript zu schreiben (Basics zu verstehen)

- Möglichkeit Files über HTTP-Server zu veröffentlichen

- Die Files, die für die offline-Ansicht aktiviert werden sollen (HTML, JavaScripte, CSS, Bilder)

Eine Web-Anwendung „Gears“-fähig machen

1. Das Gears-Browser-Plugin installieren

2. JavaScript-Funktionen bzw. -Skripte
   für die Nutzung der Gears-APIs definieren und einbinden

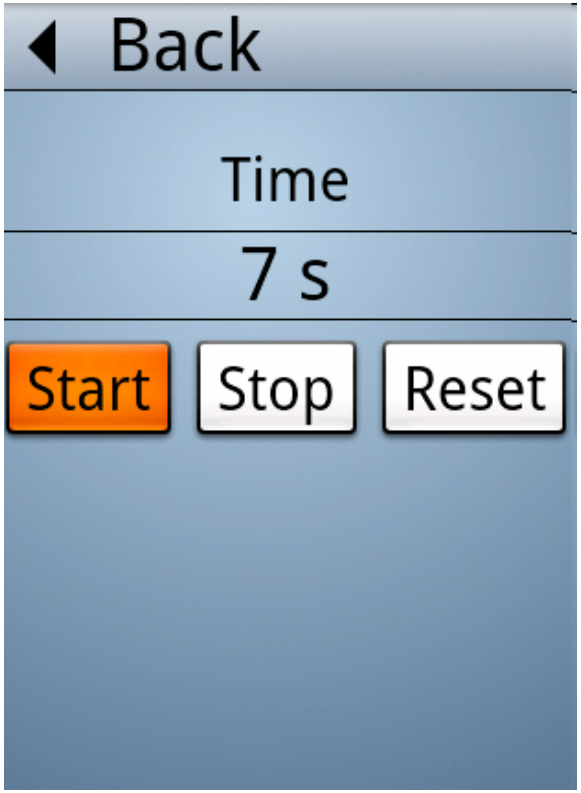
3. Manifest-File erstellen,
   das alle offline zu cachenden Files listet

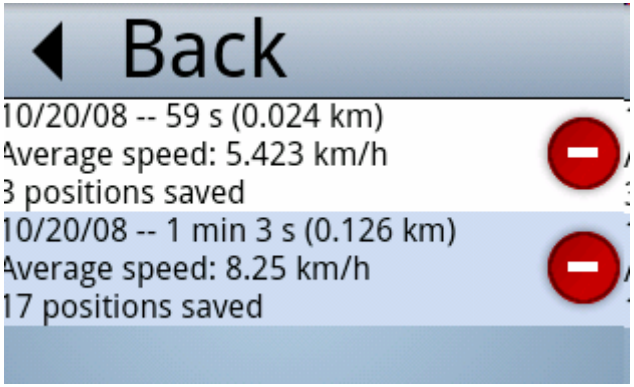
4. Alle Files auf den Online-Server uploaden

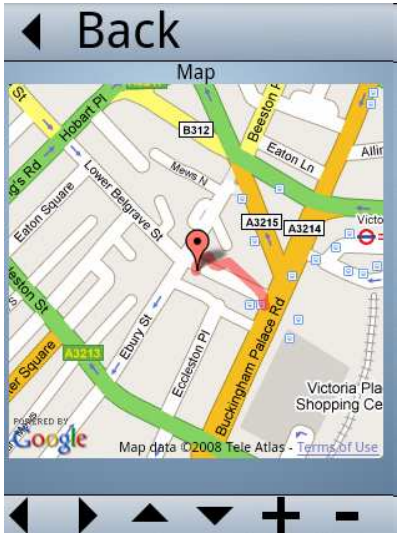
Beispiel: „RunningMan“

Eine JavaScript-Reise-Stopuhr-Applikation, die die Gears-APIs auf Android nutzt.

Sie ist ortsbestimmend und misst die Dauer, Länge und Geschwindigkeit einer Reisestrecke, deren Route anschließend auf einer Karte abgebildet werden kann.



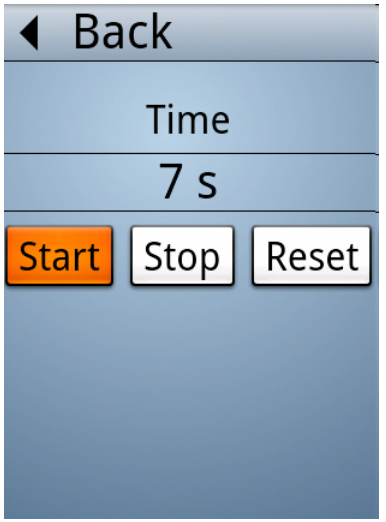




Gears on Android: „RunningMan“

Zu nutzende Gears-APIs:

- LocalServer
- Database
- Geolocation
- Desktop



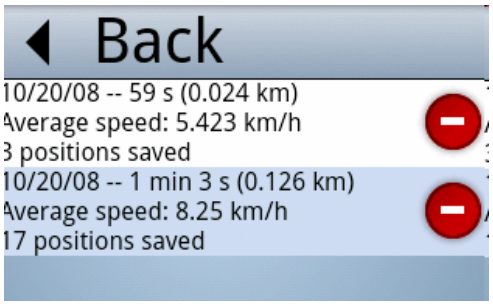
Zusätzlich noch:

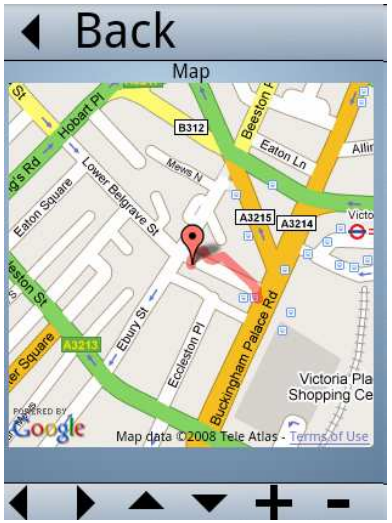
- Google Maps

(Google Maps API Key besorgen:

Hierfür wird ein Google-Account benötigt

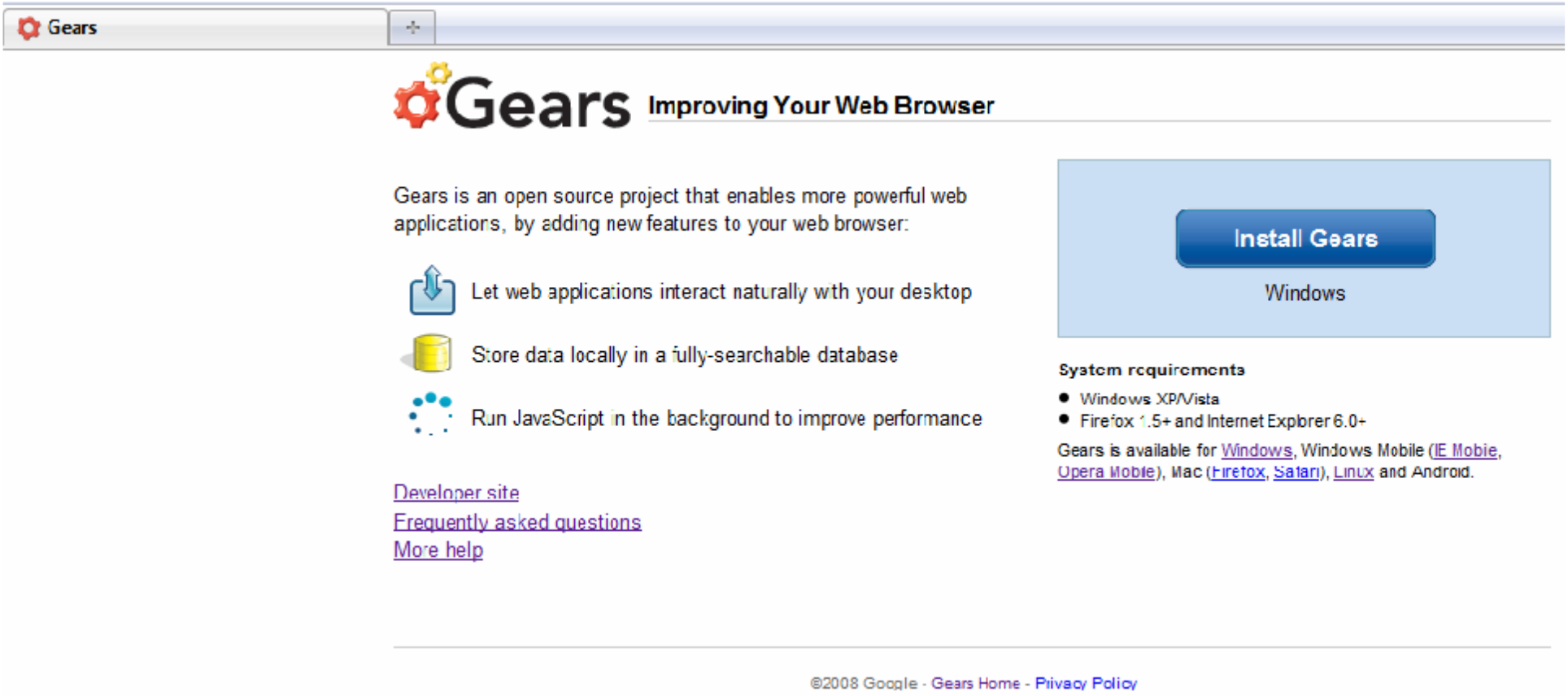
http://code.google.com/apis/maps/signup.html)





Gears on Android: RunningMan

Schritt 1: Gears installieren (http://gears.google.com)



Gears on Android: RunningMan

Schritt 2: JavaScript-Funktionen für die Nutzung der Gears-APIs „index.html“

```
<html>
<head>…
<link rel="stylesheet" type="text/css" href="../run.css" />
<script type="text/javascript" src="../../gears_init.js"></script>
<script type="text/javascript" src="model.js"></script>
</head>
<body onload="main()">
<div id="mainScreen" align="center">…
<div class="go-button" onclick="go('watch')“ >Stopwatch</div>
<div class="go-button" onclick="go('journeys')“ >Journeys</div>

<div id="watch" align="center">…
<input type="button" onclick="startRun()" value="Start"></input>
<input type="button" onclick="stopRun()" value="Stop"></input>
<input type="button" onclick="resetRun()" value="Reset"></input>
<div class="back-button" onclick="go('mainScreen')“ >Back</div>

<div id="journeys“ >…
<div class="back-button" onclick="go('mainScreen')“ >Back</div>

<div id="location" align="center">…
<div class="back-button" onclick="go('mainScreen')“ >Back</div>
…
</body>
</html>
```

JavaScripte einbinden

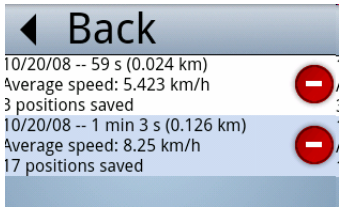
Über „main()“ selbstdefinierte JS-Funktionen für API-Verwendung aufrufen

Verwenden ebenfalls die selbst definierten JS-Funktionen für API-Verwendung

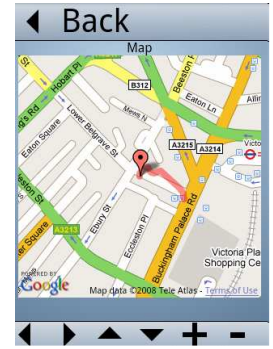
mainScreen

watch

journeys



location



Schritt 2: JavaScript-Funktionen für die Nutzung der Gears-APIs

„model.js“ – Übersicht über die wichtigsten JS-Funktionen der Anwendung

```
var global = new Object();

function main() {
global.startTime = null;
global.geo = google.gears.factory.create('beta.geolocation');
global.mapZoom = 15;
updateTime();
global.siteUrl = "http://code.google.com/apis/gears/samples/running_man/step8/index.html";
initDB();
installShortcut();
captureOffline();
loadMapsAPI();
}
```

Geolocation-Objekt zur Positionsbestimmung anlegen - Für Nutzung des Geolocation-API

initDB(); → Anlegen einer Datenbank mit Tabellen – verwendet Database-API
installShortcut(); → Anlegen eines Shortcuts – verwendet Desktop-API
captureOffline(); → Speichern der Offline-Ressourcen - verwendet LocalServer-API
loadMapsAPI(); → Laden des Maps-API – lädt das Google-Maps-API

startRun() → Startet die Stopuhr – verwendet Database-API und Geolocation-API
stopRun() → Stopt die Stopuhr – verwendet Database-API und Geolocation-API
resetRun() → Setzt die Stopuhr zurück

on_journeys() → Zeigt die Laufzeiten mit Reiseinfos an – verwendet Database-API

showMap() → Zeigt die zu einer Laufzeit gehörende Reiseroute über eine Google Map an - verwendet Database-API und Maps-API

Schritt 2: JavaScript-Funktionen für die Nutzung der Gears-APIs

„model.js“ – Database-API verwenden

```
function initDB() {
global.db = google.gears.factory.create('beta.database');
global.db.open('stopwatch');

global.db.execute('CREATE TABLE IF NOT EXISTS Preferences (Name text, Value int)');
global.db.execute('CREATE TABLE IF NOT EXISTS Times (StartDate int, StopDate int, Description text)');
global.db.execute('CREATE TABLE IF NOT EXISTS Positions
(TimeID int, Date int, Latitude float, Longitude float, Accuracy float, Altitude float, AltitudeAccuracy float)');
}
```

Datenbank anlegen

Tabellen in der Datenbank anlegen per SQL

Preferences
Name
Value

Einstellungen speichern

z.B. (‘Shortcut‘, ‘true‘)

→ Shortcut wurde schon installiert..

Times
StartDate
StopDate
Description

Laufzeiten Speichern mit Startzeit, Stoppzeit und Dauer

Positions
TimeID
Date
Latitude
Longitude
Accuracy
Altitude
AltitudeAccuracy

Speichern von Positionsangaben

Gears on Android: RunningMan

Schritt 2: JavaScript-Funktionen für die Nutzung der Gears-APIs

„model.js“ – LocalServer-API verwenden

```
function captureOffline() {
global.localserver = google.gears.factory.create("beta.localserver");

global.store = global.localserver.createManagedStore("Stopwatch");

global.store.manifestUrl = "manifest.json";

global.store.checkForUpdate();
}
```

LocalServer-Objekt anlegen

ManagedResourceStore anlegen

Manifest-File dem ManagedResourceStore zuweisen

(Das Manifest-File muss diesen Namen tragen)

Die im Manifest-File gelisteten Files im ManagedResourceStore speichern

Gears on Android: RunningMan

Schritt 2: JavaScript-Funktionen für die Nutzung der Gears-APIs

„model.js“ – Desktop-API verwenden

```
function installShortcut() {
if (getPreference('Shortcut') == false) {

var desktop = google.gears.factory.create('beta.desktop');
desktop.createShortcut("RunningMan", global.siteUrl,
{ "48x48" : "../images/icon.png" }, "RunningMan Application");

setPreference('Shortcut', true);}
}
```

Überprüft in Preferences-Tabelle, ob Shortcut schon installiert wurde

Desktop-Objekt anlegen

Über Desktop-API einen Shortcut „RunningMan“ erzeugen

Speichert in Preferences-Tabelle, dass Shortcut installiert wurde

Gears on Android: RunningMan

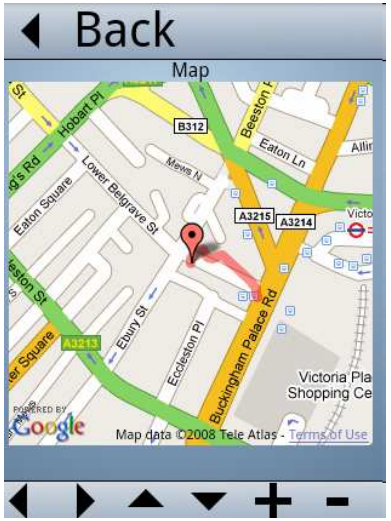
Schritt 2: JavaScript-Funktionen für die Nutzung der Gears-APIs

„model.js“ – Google-Maps-API laden

Funktion zum Laden des Google-Maps-API.

```
function loadMapsAPI() {
var script = document.createElement("script");

var key =
'ABQIAAAAsZ2C8blcRF_NO1nRClSs4xT7_rQBWrRQBxIxHmD38f4UhmkYzRT_oAn4JlEtnpyS2NUN4Zh0L5PgQ';

script.type = "text/javascript";

script.src = "http://maps.google.com/maps?file=api&v=2&async=2&key=" +key;

document.body.appendChild(script);
}
```

Maps-API-Key



Gears on Android: RunningMan

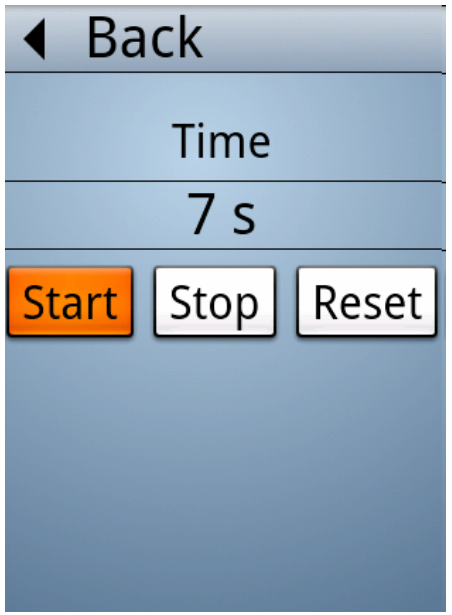
Schritt 2: JavaScript-Funktionen für die Nutzung der Gears-APIs

„model.js“ – Stopuhr-Funktionen

Startdatum mit Zeit in der Times-Tabelle speichern

GeoWatcher zum erkennen von Positionsänderungen starten und die neuen Positionen in Positions-Tabelle speichern

```
function startRun() {
global.updateInterval = setInterval("updateTime()", 1000);
global.startTime = new Date();
var time = global.startTime.getTime();
global.db.execute('INSERT INTO Times (StartDate) VALUES (?)', [time]);
global.currentTimeID = global.db.lastInsertRowId;
global.currentGeoWatcher = global.geo.watchPosition(function (position) {
global.db.execute('INSERT INTO Positions (TimeID, Date, Latitude, Longitude,
Accuracy, Altitude, AltitudeAccuracy) VALUES (?, ?, ?, ?, ?, ?, ?)',
[global.currentTimeID, position.timestamp.getTime(),
position.latitude, position.longitude, position.accuracy,
position.altitude, position.altitudeAccuracy]);
}, null, { "enableHighAccuracy" : true});}
```



Stopdatum und Zeit in Times-Tabelle speichern

```
function stopRun() {
clearInterval(global.updateInterval);
var stopDate = new Date();
var time = stopDate.getTime();
global.db.execute('UPDATE Times SET StopDate = (?) ' +
'WHERE ROWID = (?)', [time, global.currentTimeID]);
if (global.currentGeoWatcher != null) {
global.geo.clearWatch(global.currentGeoWatcher);
global.currentGeoWatcher = null; }}
```

GeoWatcher stoppen

```
function resetRun() {
stopRun();  global.startTime = null;  updateTime(); }
```

Gears on Android: RunningMan

Schritt 2: JavaScript-Funktionen für die Nutzung der Gears-APIs

„model.js“ – Journeys anzeigen

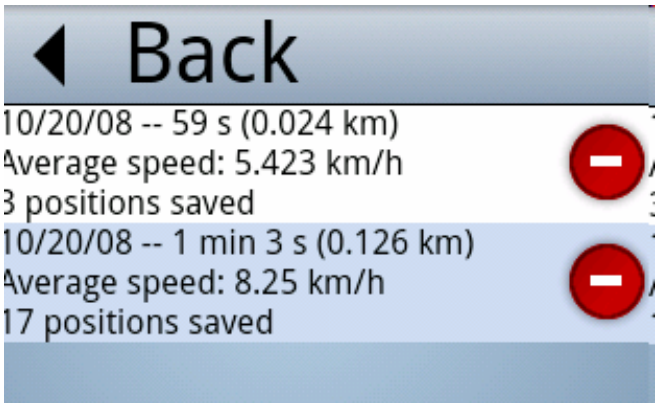
```
function on_journeys() {
var rows = 0;
var html = "";
var rs = global.db.execute('SELECT ROWID, Description FROM Times');
while (rs.isValidRow()) {
var rowID = rs.field(0);
var description = rs.field(1);
if (description == null) {
description = createDescription(rowID) ;}
var rowType;
if (rows % 2 == 0) { rowType = "rowA";}
else { rowType = "rowB";}
rows++;
html += "<div class='" + rowType + "'>";
html += "<div class='rowDesc' onclick='showMap(" + rowID + ")'>"
        + description + "</div>";
html += "<div class='rowImg' onclick='deleteRecord(" + rowID + ")'>";
html += "<img src='../images/delete.png'></div";
html += "</div>";
rs.next();}
if (html != "") {
var elem = document.getElementById('journeysContent');
elem.innerHTML = html; }
}
```

Holt Laufzeitdauer (Description) aus Times-Tabelle

Laufzeitdauer ist vor Ausführung von „on_journeys()“ noch nicht berechnet

„createDescription()“ berechnet Laufzeitdauer, Entfernung, Geschwindigkeit, fügt sie zusammen und gibt alles als Description zurück

Beim Klicken auf einen Eintrag wird „showMap()“ ausgeführt



Gears on Android: RunningMan

Schritt 2: JavaScript-Funktionen für die Nutzung der Gears-APIs

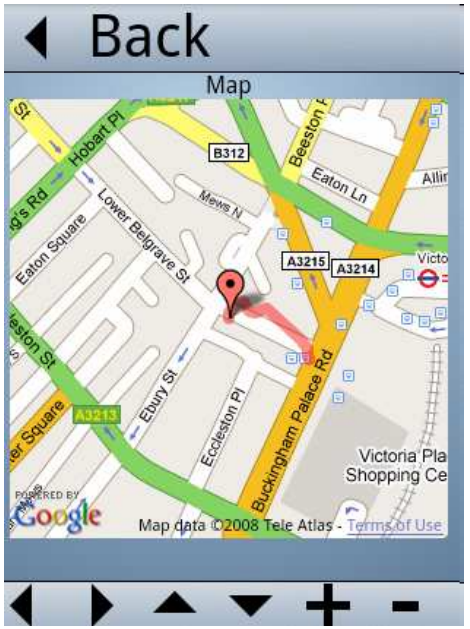
„model.js“ – Map anzeigen

Längen- und Breitengrade aus der Positions-Tabelle

```
function showMap(rowID) {
hideAllScreens();
showDiv('location');
if (window["GMap2"] == null) {
alert('No Map object!');
return; }
var rs = global.db.execute('SELECT Latitude, Longitude FROM Positions WHERE TimeID = (?) '
ORDER BY Date', [rowID]);
var lastPoint = null;
global.map = new GMap2(document.getElementById("map"));
var locations = new Array();
while (rs.isValidRow()) {
var lat = rs.field(0);
var lon = rs.field(1);
var point = new GLatLng(lat, lon);
locations[locations.length] = point;
lastPoint = point;
rs.next();
}
if (lastPoint != null) {
global.map.setCenter(lastPoint, global.mapZoom);
var polyline = new GPolyline(locations, '#ff0000', 8);
global.map.addOverlay(polyline);
global.map.addOverlay(new GMarker(point));
}}
```

Map-Positions-Punkte, aus Längen- und Breitengrad, werden in einem Array „locations“ gespeichert

Map-Positions-Punkte aus dem Array „locations“ als Routenlinie auf Map ausgeben



Gears on Android: RunningMan

Schritt 3: Manifest-File erstellen:

„manifest.json“

```
{ "betaManifestVersion": 1,
  "version": "version 1.0",
  "entries": [
                { "url": "index.html" },
                { "url": "model.js" },
               { "url": "gears_init.js" },
               { "url": "../run.css" },
               { "url": "../images/icon.png" }, { "url": "../images/button-background.png" },
               { "url": "../images/background.png" }, { "url": "../images/back-arrow.png" },
               { "url": "../images/go-arrow.png" }, { "url": "../images/delete.png" },
               { "url": "../images/left.png" }, { "url": "../images/right.png" }, { "url": "../images/down.png" },
               { "url": "../images/up.png" }, { "url": "../images/zoomIn.png" }, { "url": "../images/zoomOut.png" }
             ] }
```

Gibt die Version des Manifest-File-Formats an, diese ist entweder 1 oder 2 (Genauer Unterschied leider unklar)

Gibt die Version der Manifest-Inhalte für Aktualisierungszwecke an

Schritt 4: Files auf den HTTP-Server uploaden

(Manifest-File und alle darin gelisteten Files)

Anmerkung: Update des Manifest-File

Werden Files aktualisiert, hinzugefügt oder entfernt

→ Manifest-File updaten!
Durch Änderung des Versions-String im Manifest-File

→ Gears erfasst neuen Versions-String im Manifest und kopiert die geupdateten Inhalte automatisch auf lokalen Datenträger, wenn Site wieder online besucht wird.

```
{ "betaManifestVersion": 1,
  "version": "version 1.0",
  "entries": [
              { "url": "index.html" },
              { "url": "model.js" },
             { "url": "gears_init.js" },
             { "url": "../run.css" },
             { "url": "../images/icon.png" }, { "url": "../images/button-background.png" },
             { "url": "../images/background.png" }, { "url": "../images/back-arrow.png" },
             { "url": "../images/go-arrow.png" }, { "url": "../images/delete.png" },
             { "url": "../images/left.png" }, { "url": "../images/right.png" }, { "url": "../images/down.png" },
             { "url": "../images/up.png" }, { "url": "../images/zoomIn.png" }, { "url": "../images/zoomOut.png" }
           ] }
```

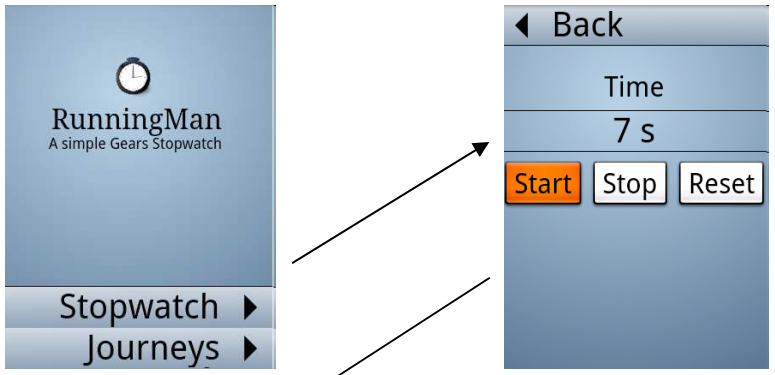
z.B. “version 1.1“

Gears on Android: RunningMan

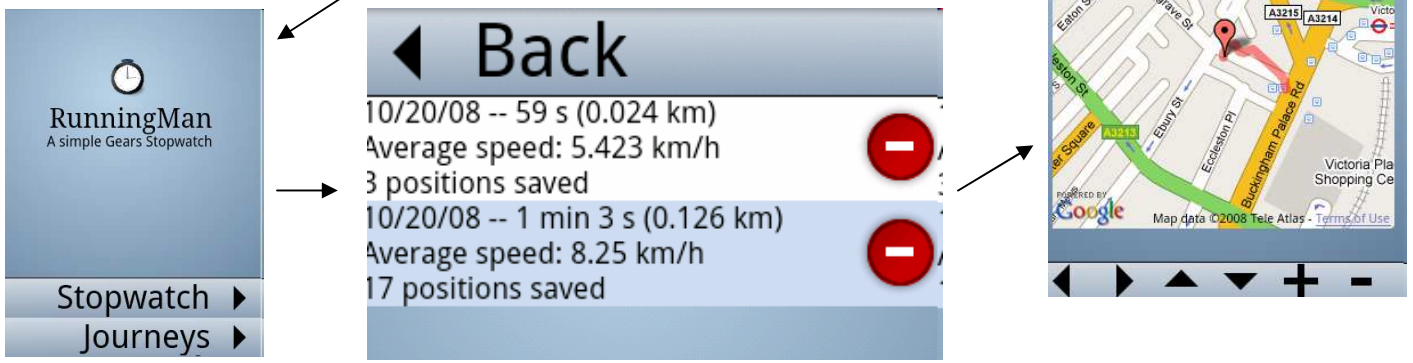
Szenario

1. Shortcut

2. Reise-Stopuhr



3. Reise-Infos



Sicherheit bei Gears

- **“Same-origin policy”- Sicherheitskonzept:**
  Eine Webseite kann nur auf Ressourcen der gleichen Herkunfts-Quelle zugreifen.
  Für eine Gears-Seite bedeutet dies:
  Database: Kann immer nur die Datenbank der eigenen Gears-Seite öffnen
  LocalServer: Kann immer nur die URLs der eigenen Gears-Seite speichern

- **Warndialog:**
  Beim ersten Besuch einer Gears-Seite wird ein Warndialog angezeigt, dass die Seite Gears verwenden möchte.
  Dieser Warndialog muss erst bestätigt werden, damit die Seite die Gears-APIs verwenden kann.

- **End-Benutzer Daten**:
  Gears-Daten sind nur über das jeweilige Benutzer-Profil des Betriebssystems zugänglich bzw. über dessen Login

Risiko bei Gears

- **Gleicher OS-Login:**
  Benutzen zwei Personen den selben Login für ein Betriebssystem, so können sie gegenseitig auf ihre gespeicherten Gears-Daten zugreifen.
  (z.B. Problematisch in Internetcafes, wenn Gears-Anwendungen das Speichern, Ändern und/oder Abrufen von persönlichen Daten ermöglichen)

- **SQL-Injection-Angriffe:**
  Maskiert ein Gears-Entwickler die Benutzereingaben in SQL-Statements nicht ausreichend, besteht die Gefahr, dass Daten über eingeschleuste SQL-Befehle ausspioniert oder verändert werden können.
  Tipp: Benutzereingaben beim Programmieren von SQL-Statements nur über (?)-Austausch-Parameter übergeben.
  Beispiel:

```
db.execute('insert into MyTable values (' + data + ')');    falsch
db.execute('insert into MyTable values (?)', data);         richtig
```

Anmerkung:
Wann und wie die Gefahr für SQL-Injection bestehen würde ist leider auf den Google Gears Seiten nicht erläutert → Fragestellung: Wie ist SQL-Injection möglich bei “same-origin policy”?

Ausblick: HTML5 vs. Gears

- Viele Funktionen von Gears sind schon in der Spezifikation von HTML5 enthalten
- Google empfiehlt Entwicklern, die Nutzung von HTML5 statt Gears!
- Google entwickelt Gears zwar nicht mehr weiter, stellt es aber auch nicht ein!
  Dadurch verlieren Seiten, die Gears verwenden nicht ihre Funktionen.

HTML5	Gears
- Webstandard	- Plugin
- für alle (neueren) Browser, jedoch mit funktionellen Einschränkungen	- nur für ältere oder ausgewählte Browser
- wird laufend weiterentwickelt	- wird funktionell nicht mehr weiter entwickelt

Aktuelle Beispiele, die besonders für **HTML5** sprechen:
- Die neue Version vom „Google Chrome“-Browser unterstützt HTML5 soweit, dass Gears nicht mehr notwendig ist!
- Es wird Keine Gears-Versionen mehr für einige neuere Plattformen geben, wie z.B. den Safari-Browser unter Mac OS X 10.6 (Snow Leopard).

Aktuelle Beispiele, die noch für **Gears** sprechen:
- Der aktuelle IE und Firefox werden mit Gears-Versionen unterstützt (z.B. FF 3.6)
- Um HTML5 zu nutzen müssen Entwickler ihre Gears-Anwendungen umständlich portieren

Ausblick: HTML5 vs. Gears

HTML5 Spezifikation	Gears Plugin	IE 6&7	IE 8	FF3	FF 3.5	Safari 4	Safari Mobile	Chrome	Opera
Offline	■			■	■	■	■	■	
Worker	■				■	■		■	
Local/ Session Storage			■		■	■	■		
Local Database	■					■	■	■	
Geolocation	■				■		■	■	
Video/Audio					■	■	■	■	■
Native JSON			■		■				
Cross Site XHR			■		■	■	■	■	
Cross-document messaging			■	■	■	■	■	■	
Canvas				■	■	■	■	■	■
DOM Selectors			■		■	■	■	■	
CSS Transform (Draft)					■	■	■	■	
CSS Media Queries					■	■	■	■	■
Drag&Drop vom Desktop	■								
Drag&Drop (HTML5)					■	■		■	
Blob Builder/ File Upload with XHR	■				■				

Zusammenfassung

Was haben wir gelernt?

- Gears ist ein Browser-Plugin, das die Nutzung verschiedener JS-APIs ermöglicht, über die die wichtigsten Module für Offline-Funktionalität bereitgestellt werden

- Nicht jede Webanwendung ist automatisch durch Installation des Gears-Plugins offline nutzbar

- Web-Anwendungen werden erst durch das Programmieren von JS-Funktionen, die auf die Gears-APIs zugreifen gearsfähig

- Gears wird zwar offiziell nicht mehr weiterentwickelt, ist dafür aber open-source und für ausgewählte (ältere) Browser immernoch sinnvoll

Gears

Puh! Endlich ENDE..

oder habt ihr noch Fragen?..

die ich auch beantworten kann ;-)

Literatur:

Gears-Homepage: http://gears.google.com/

Gears-Website-Tutorial:
http://code.google.com/intl/de-DE/apis/gears/tutorial.html

Gears-RunningMan-Tutorial:
http://code.google.com/intl/de-DE/apis/gears/articles/running_man.html

Gears-Artikel:
http://www.linux-magazin.de/Online-Artikel/Anwendungen-fuer-Online-und-Offline-mit-Google-Gears-Teil-1/%28offset%29/4
http://www.puremedia-online.de/fileadmin/Publikationen/2008-04-17_web20kongress_offline.pdf

Gears-Blob-Beispiel: http://code.google.com/p/gears/issues/detail?id=994

Gears vs HTML5 -Artikel:
http://www.golem.de/0912/71588.html
http://www.pcwelt.de/start/dsl_voip/online/news/2106837/google-setzt-auf-html5-statt-google-gears/
http://almaer.com/blog/gears-as-a-bleeding-edge-html-5-implementation
http://www.pocketbrain.de/newsticker/news/2713-google-html5-statt-gears.html
http://spreadsheets.google.com/ccc?key=rp3FEESll4-WG1CwPL6hpPw&hl=en
http://www.golem.de/1002/73274.html
http://gearsblog.blogspot.com/2010/02/hello-html5.html