# INFORMATIONSINTEGRATION UND WEBPORTALE

**Vorlesung #5, 24.11.2014**
Mobile WebApps: Frameworks und Architekturen

**Verónica Rivera-Pelayo**
**(rivera@fzi.de)**



Klick-And-Bau

**Informationsintegration und Webportale**

# ZU MEINER PERSON

# Dipl.-Inf. Verónica Rivera-Pelayo (FZI)

- **Ausbildung**
  - Informatik (UPC-BARCELONA TECH)
- **Aktuelle Tätigkeit**
  - wiss. Mitarbeiterin im IPE am FZI
  - Doktorandin am AIFB, Prof. Rudi Studer
- **Interessen**
  - Mobile Technologien, Context Management, HCI
  - Technology Enhanced Learning, Reflektives Lernen, Learning Analytics
  - Quantified Self und Erfassung von relevanten Daten für RL
  - Wissensmanagement, Semantic Web Technologies
- **Aktuelle Projekte**
  - MIRROR – Reflective Learning at Work
- **Lehre**
  - Seit 2011 TGL Seminar im SoSe (http://tgl.fzi.de)
  - Seit 2012 IIWP

# Motivation

- **Ziel dieser Vorlesung:**

  Generelle Kenntnis der Technologien und Anforderungen zur Erstellung von mobilen nativen und Web-anwendungen

- Mobilität
- Erreichbarkeit
- Kontextualisierung
- Personalisierung…

Herausforderungen:

- Vielzahlt an Geräte und BS
- Dynamisch
- Ressourcen…

# Inhalt

## Mobile Geräte

- Technologieübersicht
- Arten mobiler Anwendungen
- Probleme und aktuelle Trends

## Entwicklung mobiler & Web Apps

- Entwicklungsansätze
- Native Entwicklung
- Cross-Plattform-Entwicklung

- Offline Modus
- Mobile Web Apps mit GWT

Photo illustration by The New York Times

# Mobile Endgeräte: die Ären


- Brick Phone – Candy Bar – Feature Phone – Smartphone – Touch Phone

- GSM, CDMA, TDMA
- Kleine Größe
- SMS Service

- GPRS, HSCSD
- Datenfähig
- Kamera & MMS
- Verbreitung

- GPRS, HSDPA Wi-Fi
- Emails Treiber
- Gadget

- GPRS, HSDPA Wi-Fi, LTE
- Sensors, MEMS
- Media Plattform
- *All about web…*

# Mobile Endgeräte: die Ären

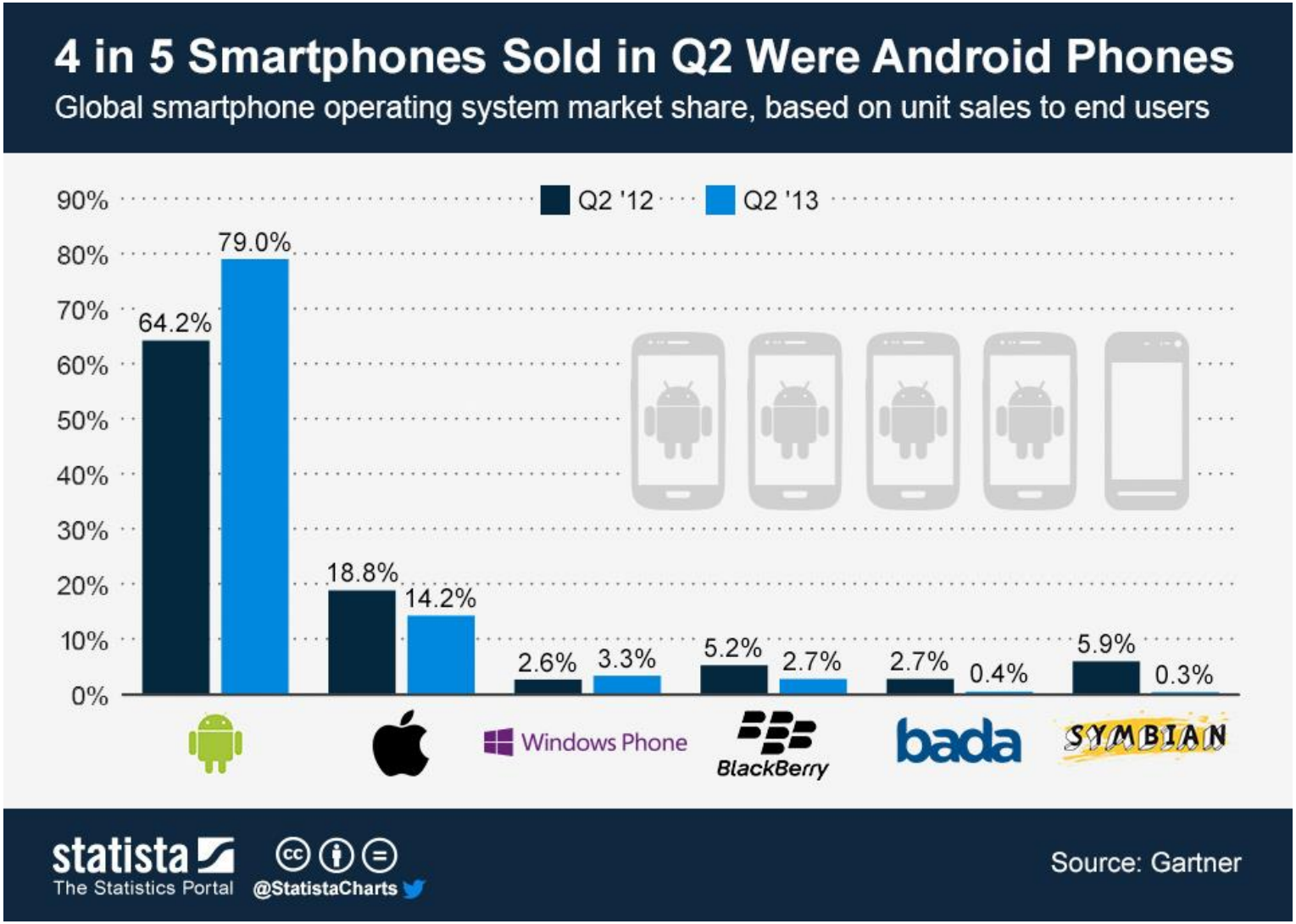

- Brick Phone – Candy Bar – Feature Phone – Smartphone – Touch Phone


- Nicht nur Handys, auch PDA, Tablets etc.

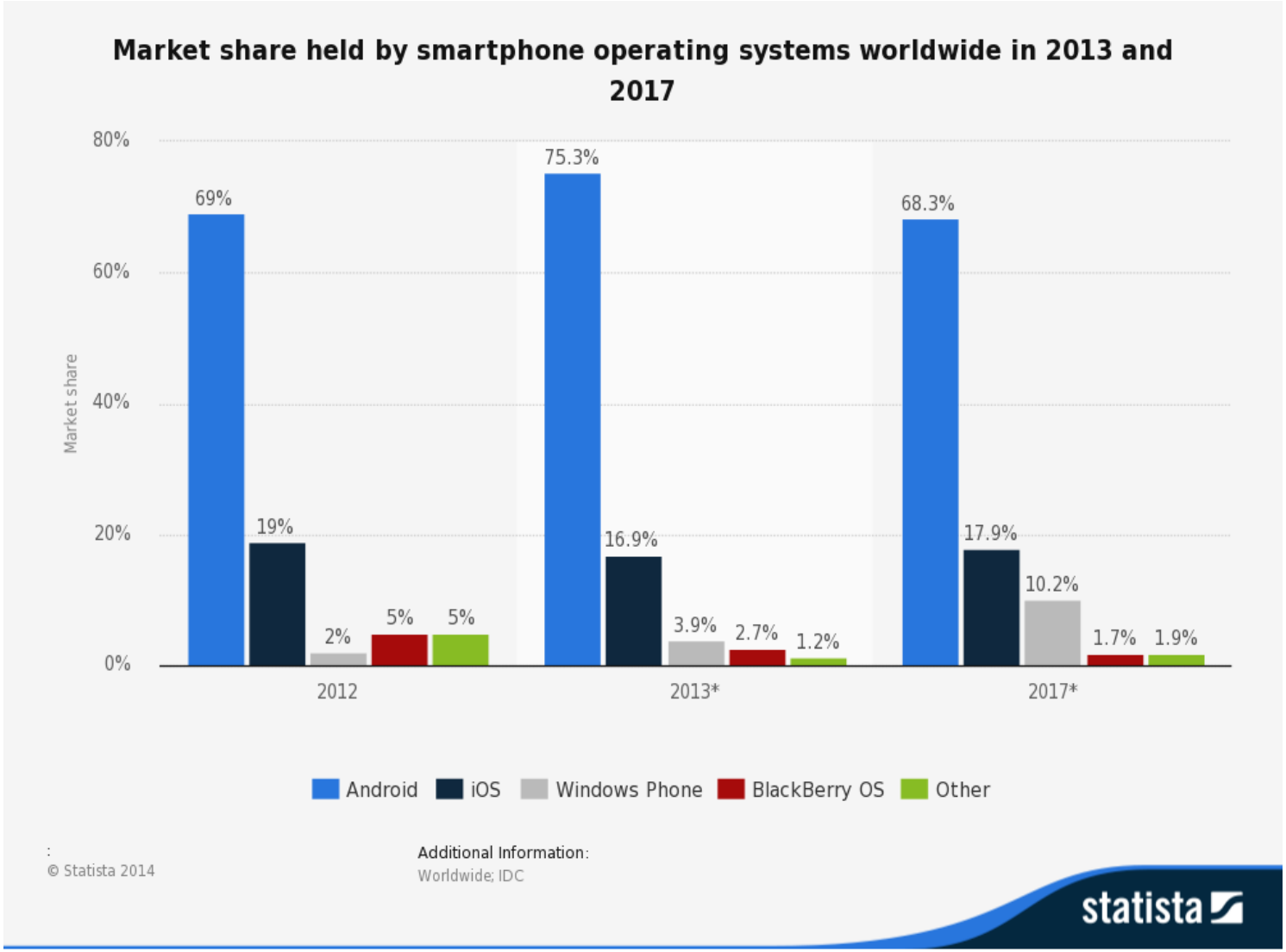
# Mobile Betriebssysteme

- Symbian (Hauptsächlich Nokia / Sony) - Eingestellt 2012
  - Ermöglicht Entwicklung nativer Apps (allerdings kompliziertes Deployment basierend auf Zertifikaten), C++
- Windows Mobile 7 / 8
  - Native Apps, Entwicklungsumgebung per .NET Framework
- WebOS (Hewlett Packard) - Eingestellt 2010
  - Native Apps, Entwicklung per C++
- Blackberry OS
  - Ausschliesslich für Blackberry Geräte, Native Apps per JAVA
- Apple iOS
  - Aussschliesslich für Apple Geräte, Entwicklung mit Objective C
- Google Android
  - Mittlerweile grösste Verbreitung über viele Hersteller, JAVA
- Bada OS (Übergang ins TIZEN OS – erstes Gerät Februar 2014)
  - Ausschließlich für Samsung Geräte, Entwicklung in C++

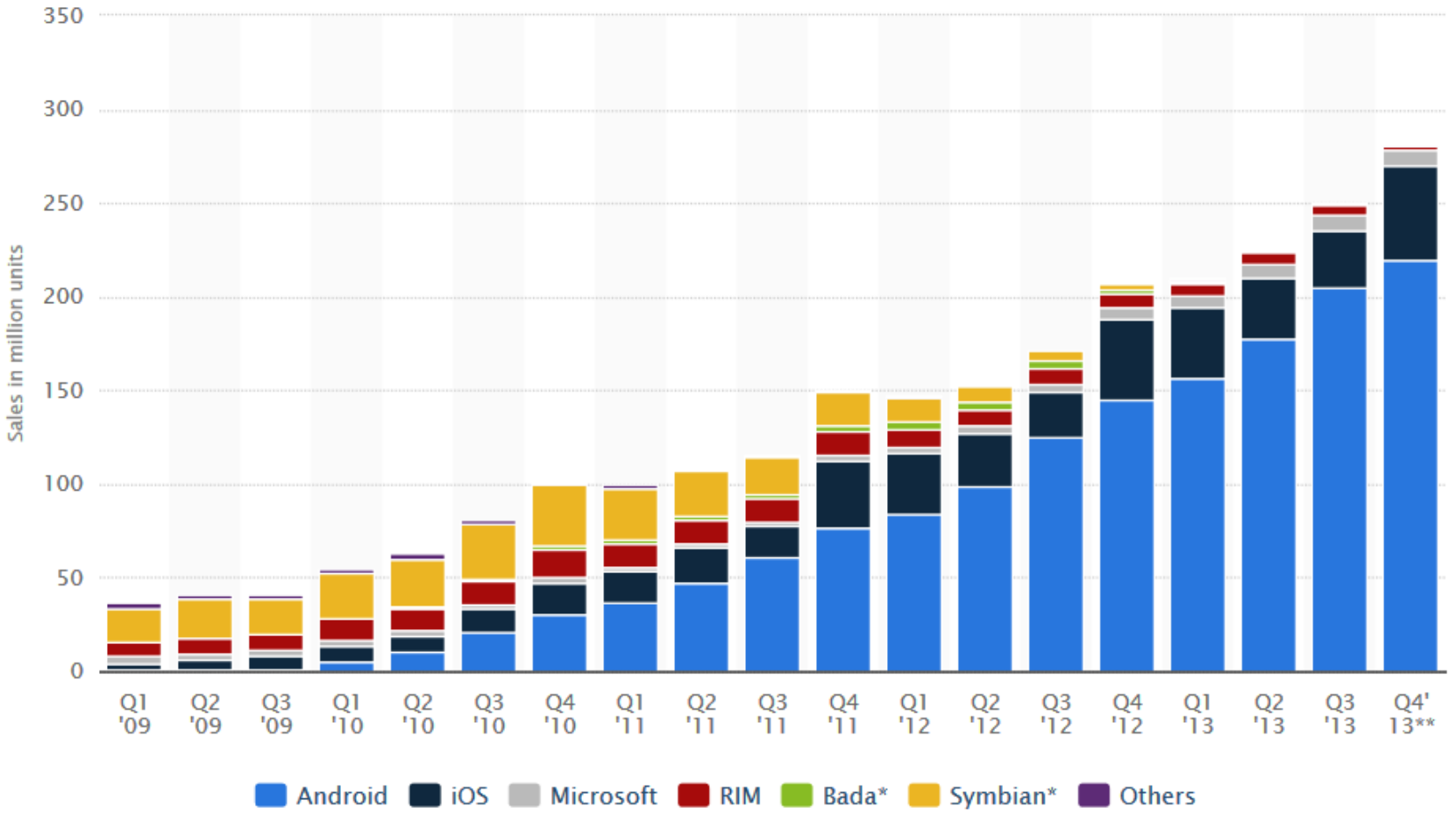
# Mobile Betriebssysteme (Verkauf in 2013)



# Mobile Betriebssysteme (Prognose für 2017)



# Globale Smartphone Absatz






http://www.statista.com/statistics/266219/global-smartphone-sales-since-1st-quarter-2009-by-operating-system/

# Herstellerspezifische Umsetzungen

**Einfaches Praxisbeispiel:** Entwicklung einer Anwendung, welche Nutzung von Calling-Card Anbietern automatisiert (teilweise grosse Ersparnis ins Ausland, früher auch zu anderen Mobilfunkanbietern Inland).

**Ablauf der Software**

<Wähle Einwahlnummer Callingcard> < Pausenzeichen > <Zielnummer>

**Umsetzung Pausenzeichen**


- Nokia: 3 x * Taste
- Siemens: „+“
- Sony: * Taste so lange drücken bis „p“ im Display erscheint
- Samsung: „w“, teilweise nicht unterstützt

⇒ Bereits eine solche einfache Software erfordert Spezifizierung auf Anbieter, teilweise auf Modellebene!

# Mobiler Webzugriff / Internetzugang

- Für „klassische Mobiltelefone“
  - iMode : proprietäre Entwicklung, Japan, teilweise durch Eplus auf europäische Geräte
  - WAP: Standard, kompaktes HTTP Protokoll, Verwendung hauptsächlich zur Übertragung von Anwendungen z.B. durch Nokia, Darstellung von Inhalten im wesentlichen per XHTML (zeitweise WML)
- Für Smartphones / Tablets
  - Mittlerweile wie bei PCs, also Firefox, Safari, Chrome, ...

## Mobiler Internetzugang

- GPRS (60 kbit) / EDGE (~ ca. 60-120 kbit) / 3G (ca. 300 kbit) / HSDPA (3-7 Mbit) / LTE ( 20-100 Mbit)
- Probleme: variiert zwischen Anbietern, je nach Region
- Datenvolumen limitiert (oft 100 – 500 MB), dann Drosselung
- Aktuelle Geräte oft auch mit WLAN

# Java 2 Micro Edition (J2ME)

Hochmotivierter Ansatz: Spezifikation für die Einheitliche Softwareentwicklung für mobile Geräte über mehrere Entwickler / Anbieter.

- Schaffung von mobiler JAVA Edition J2ME.
- Vielzahl von vorgeschlagenen Bibliotheken für u.a. Zugriff auf Kontakte, Internet, GPS, Bluetooth etc.

**Umsetzung in der Praxis :**

- Beinahe jeder Hersteller hat eigene Sicherheitszertifikate, die Zugang auf z.B. Kontakte ohne nervige Warnungen ermöglichen
- Jeder Hersteller hat Teile der Bibliotheken umgesetzt bzw. modifiziert
- Fast jede Geräteserie hatte eigene Features

⇒ Hoher Aufwand z.B. für Spieleentwicklung blieb bestehen

# Mobile App Arten

Im Kontext dieser Vorlesung auf klick-und-bau Anwendung.

Ziel: Benutzer soll mobiler Zugriff auf eine Anwendung ermöglicht werden

**Möglichkeiten**

- [SMS : z.B. Senden von Angeboten; keine Interaktion, teuer]
- Mobile Webseite
- Mobile Web Widgets
- **Mobile Web Apps**
- **Native Apps**

# Mobile Webseite

- Webseite speziell gestaltet für mobile Geräte
- Einfach in Architektur, Navigation, Design
- Gleiche Werkzeuge und Techniken wie für „normale“ Webseiten nutzbar

- Häufig rein informationell
- Fast alle Geräte können mobile Webseiten anzeigen
- Eingeschränkte *Experiences* für den Nutzer (z.B. eingeschränkte Interaktionsmöglichkeiten)
- Langsame Ladezeiten, Internetverbindung notwendig

# Mobile Webseite vs. Native App


# Mobile Web Widgets

- Kleine Anwendung, die nicht für sich selbst laufen kann, sondern auf einer *Widget* Plattform
- Auch bekannt als Gadget, Snippet, Portlet, Flake, usw.

- Leicht mit einfachem HTML, CSS und JavaScript zu erstellen
- Einfach für unterschiedliche Handsets zu deployen
- Direkter Zugriff auf Gerätefunktionalitäten und Offline-Nutzung
- Benötigt Widget Plattform auf dem Endgerät
- Umgang mit proprietären, nicht-standardisierten Techniken nötig

# Mobile Web Widgets


Quelle: Softonic

# Mobile Web App

- Laufen im mobilen Web-Browser
- Entwicklung mit Standardtechniken wie HTML, CSS, JavaScript

- Im Gegensatz zu mobilen Webseiten bieten sie dem Nutzer ein „App ähnliches“ Erlebnis
  - Z.B. mit Buttons, Echtzeitdaten, fehlender Seiten-Metapher
- Vorteile wie Mobile Webseiten, zusätzlich besseres Nutzererlebnis und Design
- Nicht immer werden alle nativen Funktionalitäten unterstützt
  - Z.B. Kamerazugriff, Offline-Modus (Speicherung von den Daten) etc.

# Mobile Web App


Quelle: bluefountainmedia.com (Tim Gray)

# Native App

- Entwickelt für eine bestimmte Plattform
- Zertifiziert
- Verkauft/verteilt über ein Betreiber-Portal oder AppStore
- Bestes Nutzererlebnis, Design, Ausnutzung der Geräte-Funktionalitäten

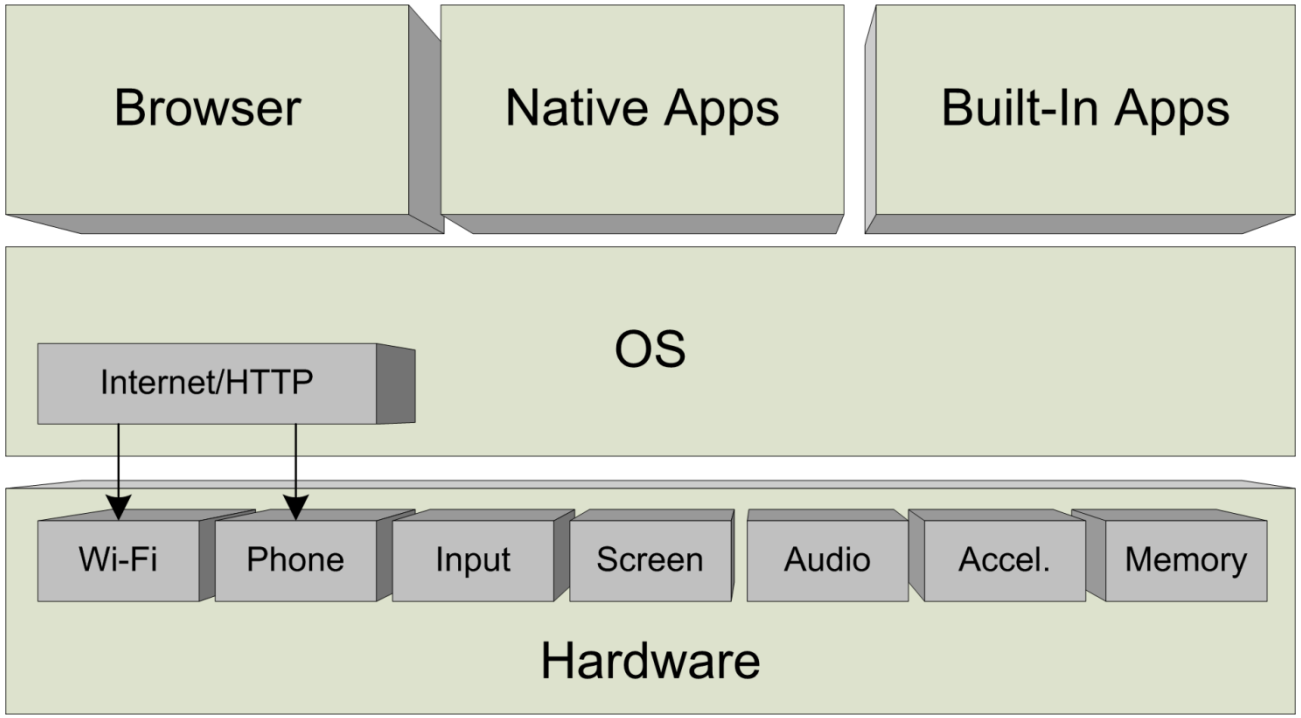
- Offline-Modus ist möglich
- Relativ einfach für eine Plattform zu entwickeln, aber kostenintensiv für mehrere Plattformen
- Zertifizierung und Verteilung über Betreiber/App Store teilweise mühsam

# Native App



Quelle: iTunes, Google Play

# Fazit




Samsung
Apple
Nokia
Blackberry
Sony
Motorola

Android
iOs
J2ME
JAVA ...

- Native App Entwicklung wünschenswert aber
  - Vielzahl an unterschiedlicher Betriebssysteme
  - Vielzahl an unterschiedlichen Geräteeigenschaften

# Trends

- Mobiltelefon: starke Fokussierung auf Touchscreens (Fullscreen)
  - Vereinheitlichung des Look & Feels bzw. der Benutzung
  - Vereinheitlichung der Ausstattung: GPS, HSDPA / LTE, Bluetooth
- Betriebssysteme: starke Fokussierung auf die „big three“
  - Android: größte Verbreitung, einheitlich per JAVA entwickelbar
  - iOS: große Verbreitung durch iPad / iPod / iPhone
  - Windows 8: bei Nokia Ersatz für Symbian

- HTML Standard
  - HTML5 verspricht per JavaScript nativen Zugriff auf Gerätefunktionen
- Cross Plattform Entwicklung
  - Entwicklungsframeworks zur Entwicklung einer Anwendung über mehrere Plattformen

# ENTWICKLUNG MOBILER WEBANWENDUNGEN

# Entwicklungsansätze

- Anforderungen:
  - möglichst breite Unterstützung mobiler Endgeräte
  - möglichst geringer Aufwand für die Implementierung für unterschiedliche Plattformen
  - Nutzung möglichst aller technischen Gegebenheiten der mobilen Geräte

- Wir können 2 unterschiedliche Entwicklungsansätze unterscheiden:
  - Native Entwicklung vs. Cross-Plattform-Entwicklung

# NATIVE ENTWICKLUNG MOBILER WEBANWENDUNGEN

# Native Entwicklung

+ sämtliche Features des jeweiligen Betriebssystems optimal unterstützt und so auch vom Entwickler nutzbar

+ es gibt einheitliche APIs zum Zugriff auf Hardware-Komponenten, wie z.B. das GPS-Modul oder die Kamera

+ Performance-Vorteil, da kein Wrapper zwischen der Software und dem Betriebssystem der Befehle umschreiben muss

→ vorausgesetzt wird eine sehr homogene Nutzerbasis

- schwierig: solange auf absehbare Zeit eine starke Fragmentierung der Märkte da ist

- in Wiederspruch zu „bring-your-own-device“

- Betriebssysteme: iOS, Windows 8, **Android**

# Native Entwicklung

- iOS
  - Sehr restriktive Hersteller-Politik von Apple:
    - Anwendungsentwicklung nur mit Apple-Rechnern mit Mac OS X
    - Umfangreiche Lizenzbedingungen zu akzeptieren
    - Entwickler muss für 99$/Jahr am iOS Developer Program teilnehmen um Zugriff auf Tools wie XCode IDE oder iOS Simulator zu erhalten
  - Entwicklung in XCode IDE mit OO-Programmiersprache Objective C
  - Nicht-Apple-IDEs: z.B. die AppCode Objective-C IDE von JetBrains

# Native Entwicklung

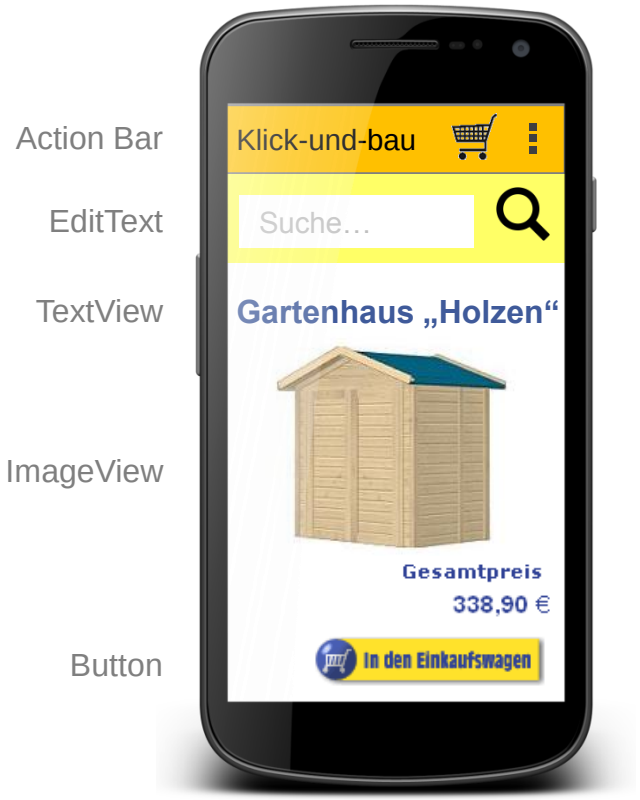
- Windows 8
  - Neue Art von Anwendung eingeführt: die Windows Store-App
    - neues Erscheinungsbild
    - auf unterschiedlichen Windows Geräten verwendbar
  - Werden über den Windows Store verkauft
  - Verschiedene Sprachen:
    - JavaScript und HTML
    - C# oder Visual Basic und XAML
    - C++ und XAML
    - C++ und DirectX
  - Sehr restriktive Hersteller-Politik von Windows:
    - Anwendungsentwicklung nur mit Windows 8 + Tools und SDK
    - Entwicklungslizenz (kostenlos) + Store-Entwicklerkonto ($19 / 14 €)
    - Lizenz wird pro Computer und für einen festen Zeitraum bereitgestellt

# Native Entwicklung

- Android
  - Entwicklung einfacher als für iOS
  - Entwicklungsumgebung Eclipse mit Plugin des Android SDK
  - Debuggen und Einbindung in einen Simulator sehr einfach möglich
  - Auch für andere Entwicklungsumgebungen gibt es entsprechende Plugins, so z.B. für Netbeans.
  - Programmiersprache Java; Anwendungen laufen in der Android-eigenen Java Virtual Machine (Dalvik Virtual Machine)
  - Android SDK enthält viele Entwicklertools u.a.:
    - Emulator
    - Debug-Monitor für die Dalvik Virtual Machine
    - Kommandozeilenanwendung Android Debug Bridge
    - Editor für die in den Geräten bzw. im Emulator vorhandene SQLite-Datenbank

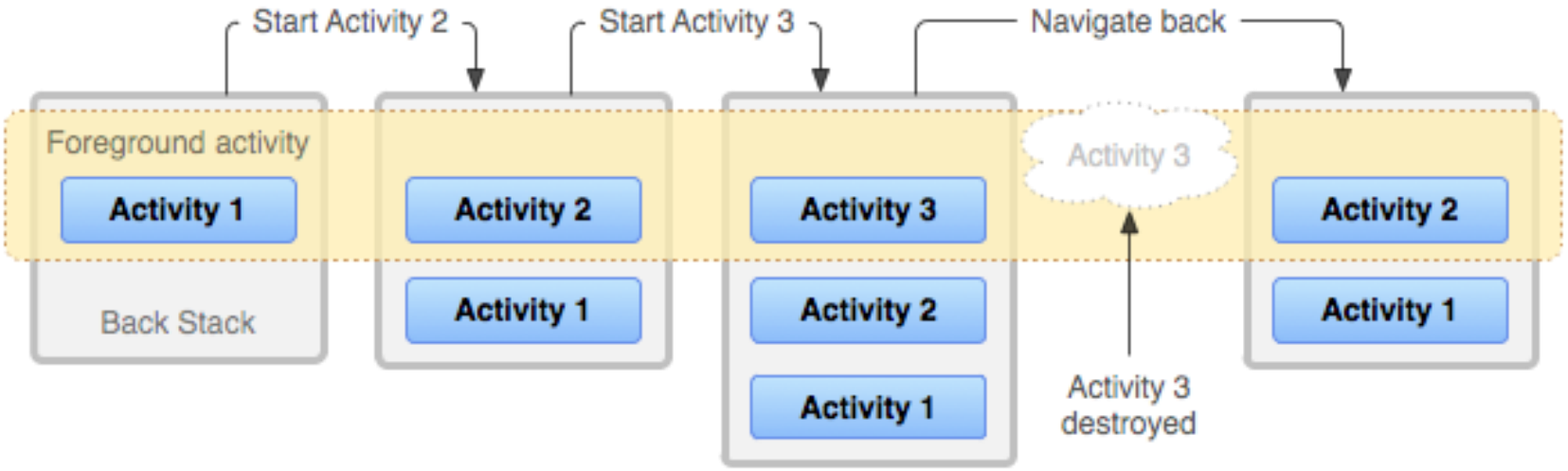
# Native Entwicklung mit Android

- Aktivitäten (Activity)
  - Ein individueller Benutzeroberflächen Bildschirm in einer Android-Anwendung
  - Visuelle Elemente, sogenannte *Views* (Komponente) können platziert werden
  - Der Benutzer kann verschiedene Aktionen durch Interaktion durchführen



# Native Entwicklung mit Android

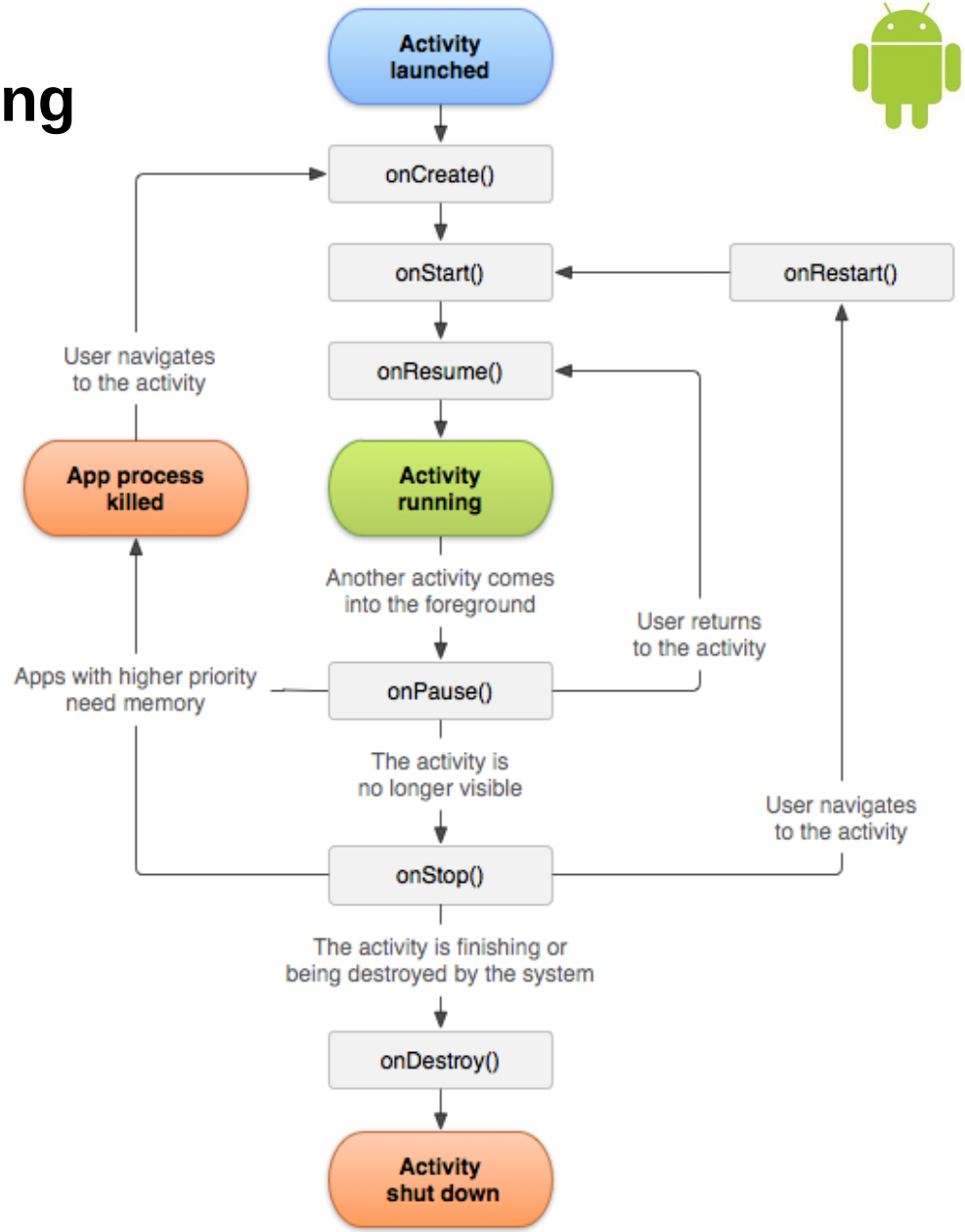
- Aktivitäten und Navigation in der Anwendung
  - Jede neue Aktivität fügt ein Element in den *back stack* ein (oben)
  - Die vorherige Aktivität bleibt in dem Stack aber ist gestoppt
  - Wenn der Nutzer den Back-Button drückt:
    - die aktuelle Aktivität wird zerstört (*destroyed*)
    - die vorherige Aktivität wird fortgesetzt



# Native Entwicklung

- Android

Activity Lifecycle

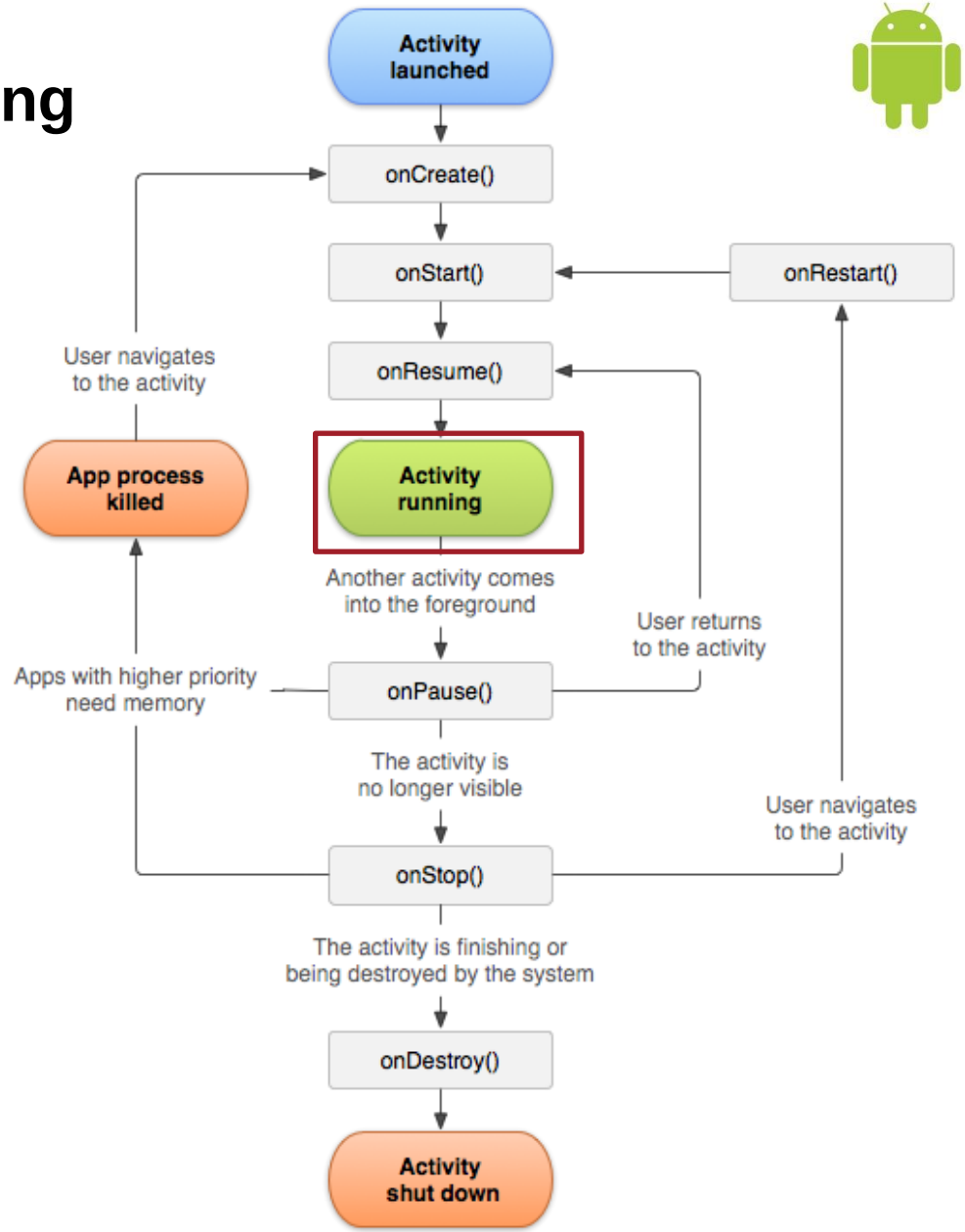


# Native Entwicklung

- Android

Activity Lifecycle

(1) RESUMED Vordergrund des Bildschirms - oben auf dem Stack

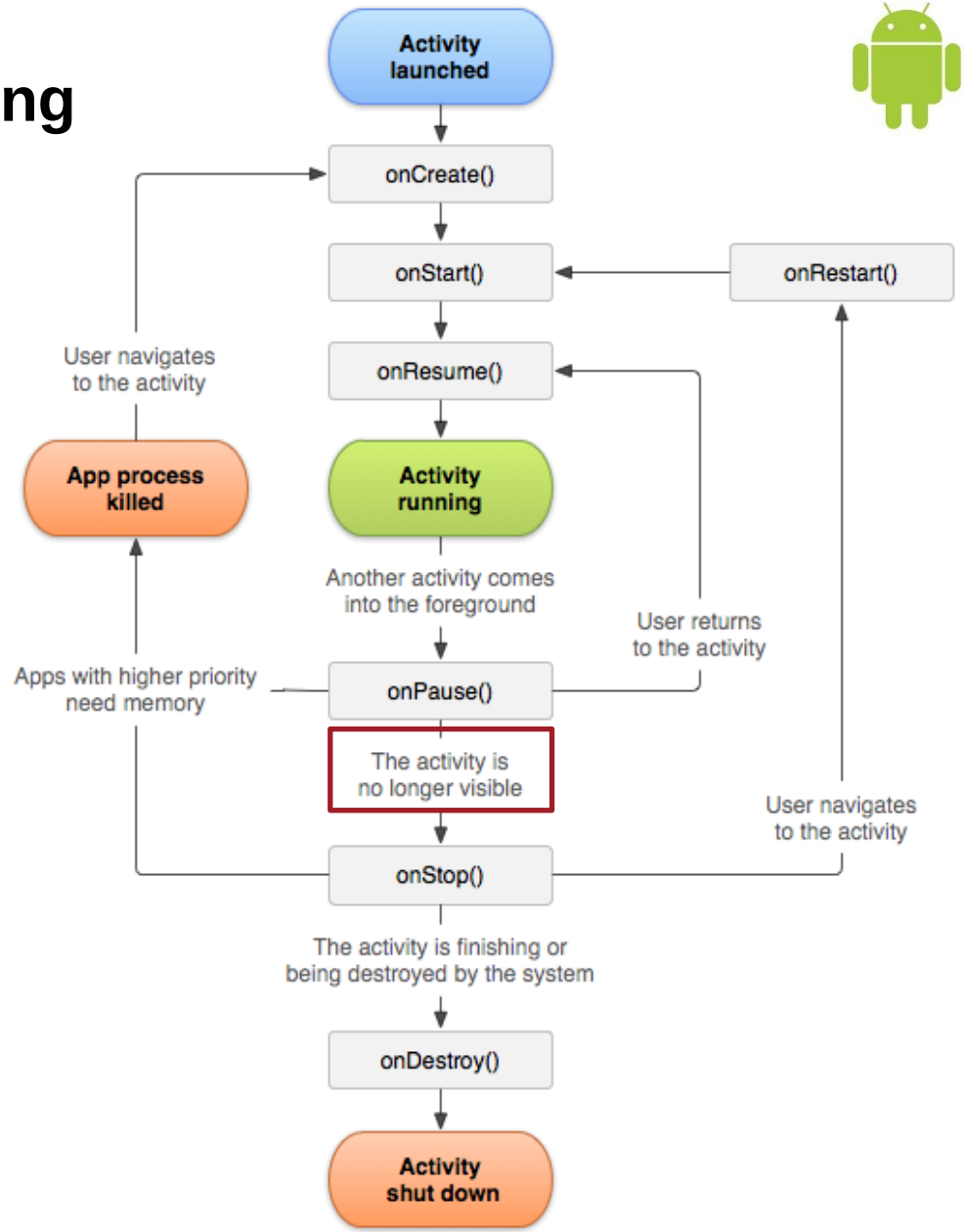


# Native Entwicklung

- Android

Activity Lifecycle

(2) PAUSED
Noch sichtbar, aber hat den Fokus verloren

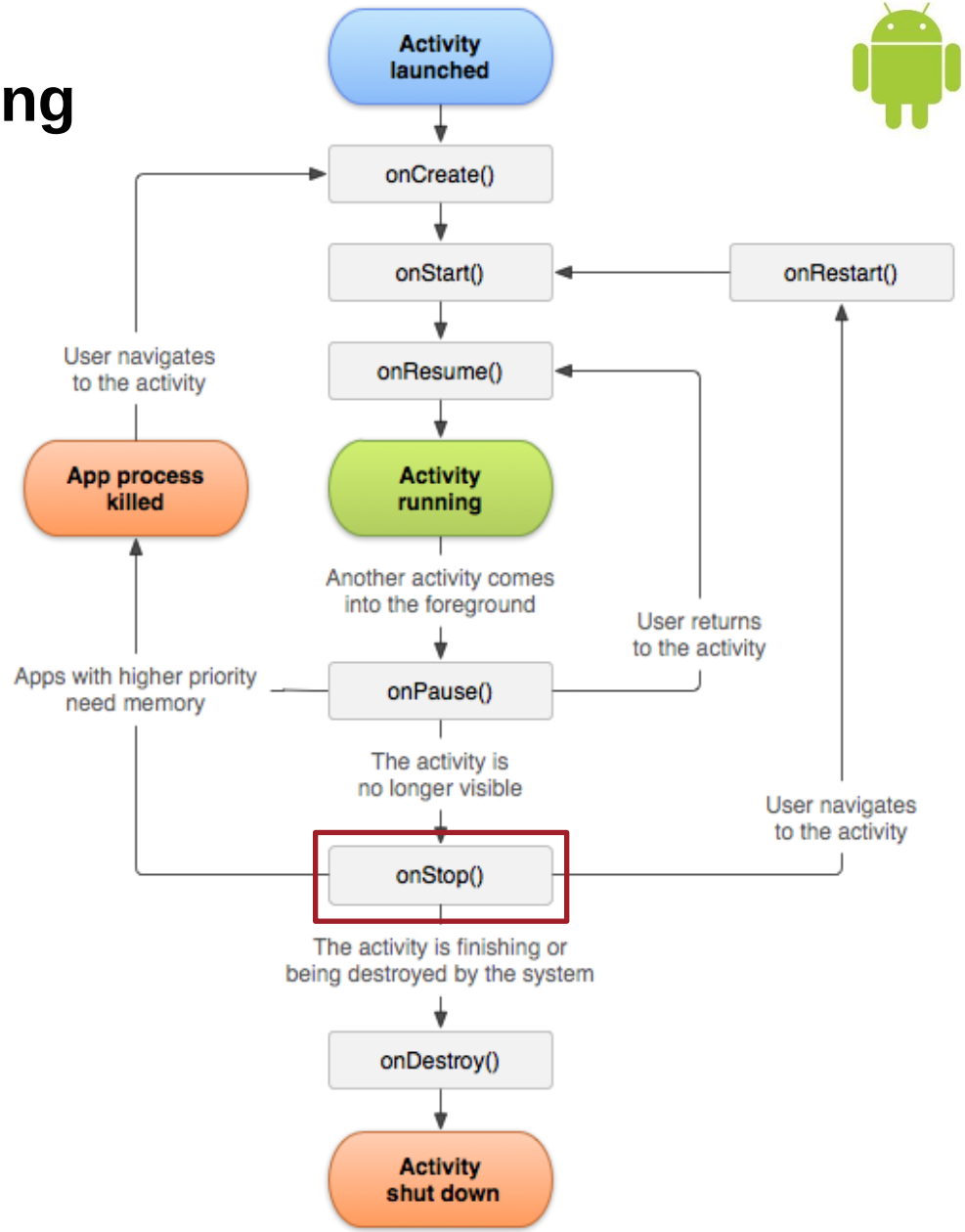


# Native Entwicklung

- Android

Activity Lifecycle

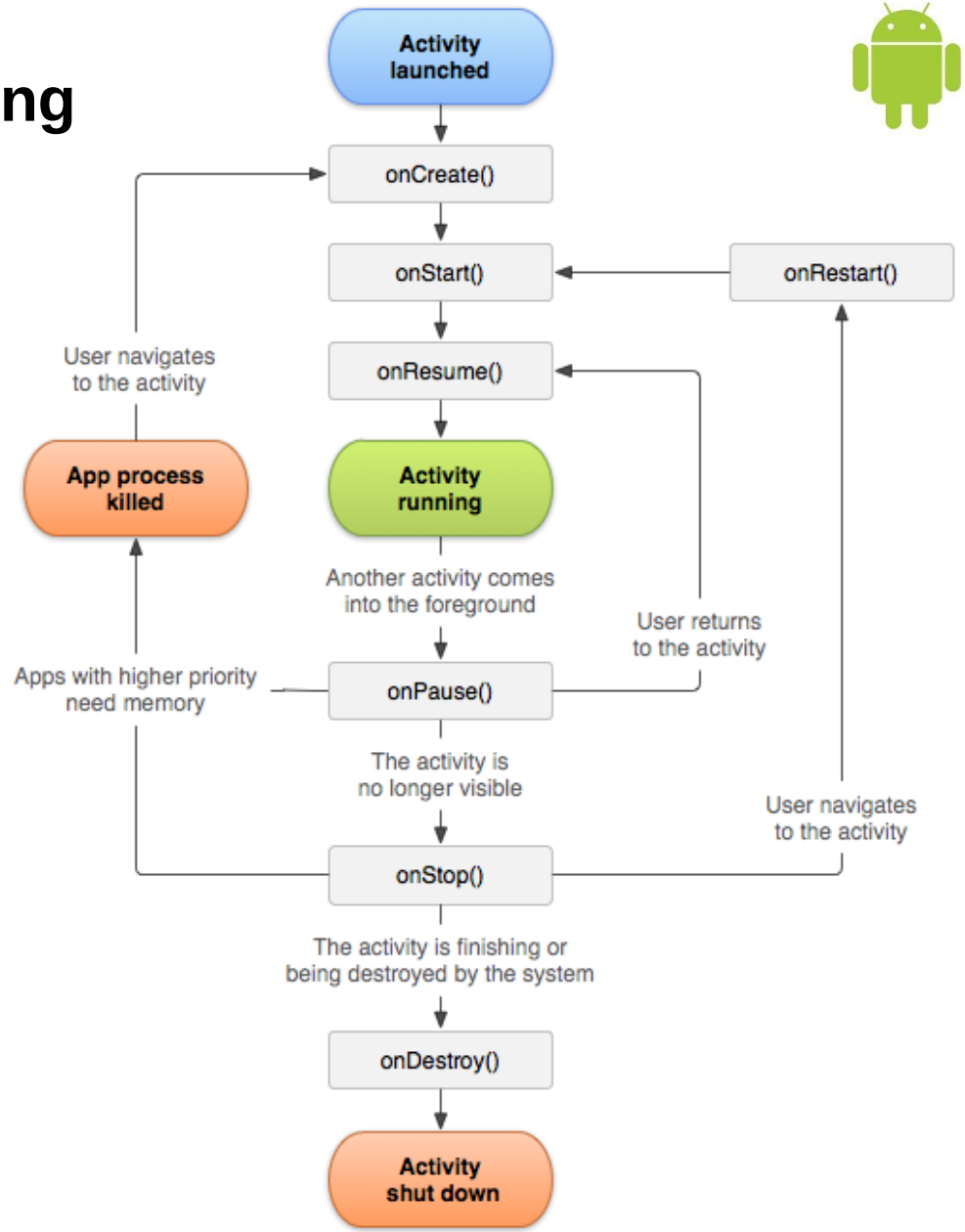
(3) STOPPED
Verdeckt von einer anderen Aktivität



# Native Entwicklung

- Android

Activity Lifecycle



# CROSS-PLATTFORM ENTWICKLUNG MOBILER WEBANWENDUNGEN

# Cross-Plattform-Entwicklung

- Gleiche Codebasis für möglichst viele Plattformen
  → Spart Kosten und Zeit bei der Entwicklung
- Aufbauend auf einer Kombination aus HTML5 und JavaScript

- Zwei OpenSource Entwicklungsmöglichkeiten stechen hervor:
  - Appcelerator Titanium
  - PhoneGap
- Für die Entwicklung der Benutzeroberflächen für PhoneGap:
  - Sencha Touch
  - jQuery mobile

# Appcelerator Titanium

- Unterstützung von iOS und Android
- Privater und kommerzieller Gebrauch unter Apache Lizenz 2.0 möglich

- Ansatz ist das Wrappen von nativen Funktionen der Betriebssysteme auf JavaScript-Funktionen
- Mit einer JavaScript-API kann man auf native Funktionalitäten der Betriebssysteme zugreifen
- Während des Build-Vorgangs wird der Code in nativen Code überführt
  - → am Ende kompilierter nativer Code auf dem Gerät
  - → Hohe Performance bei der Code-Ausführung
- Der Speicherverbrauch liegt höher als bei nativen Apps, was unter anderem an Speicherlecks im generierten Binärcode und in den zusätzlich benötigten Bibliotheken liegt

# Appcelerator Titanium

- Titanium Studio als eigene IDE
  - Hierfür Lizenz notwendig
  - Codevervollständigung erleichtert Umgang mit der Titanium-API
  - Bisher kein Debugger vorhanden, aktuell ist die Fehlersuche nur über Konsolenausgaben möglich.
- Für die Oberflächengestaltung wird auf JavaScript Stylesheets (JSS) gesetzt
  - Orientiert sich stark an CSS, unterstützt allerdings noch nicht den vollen Umfang von CSS

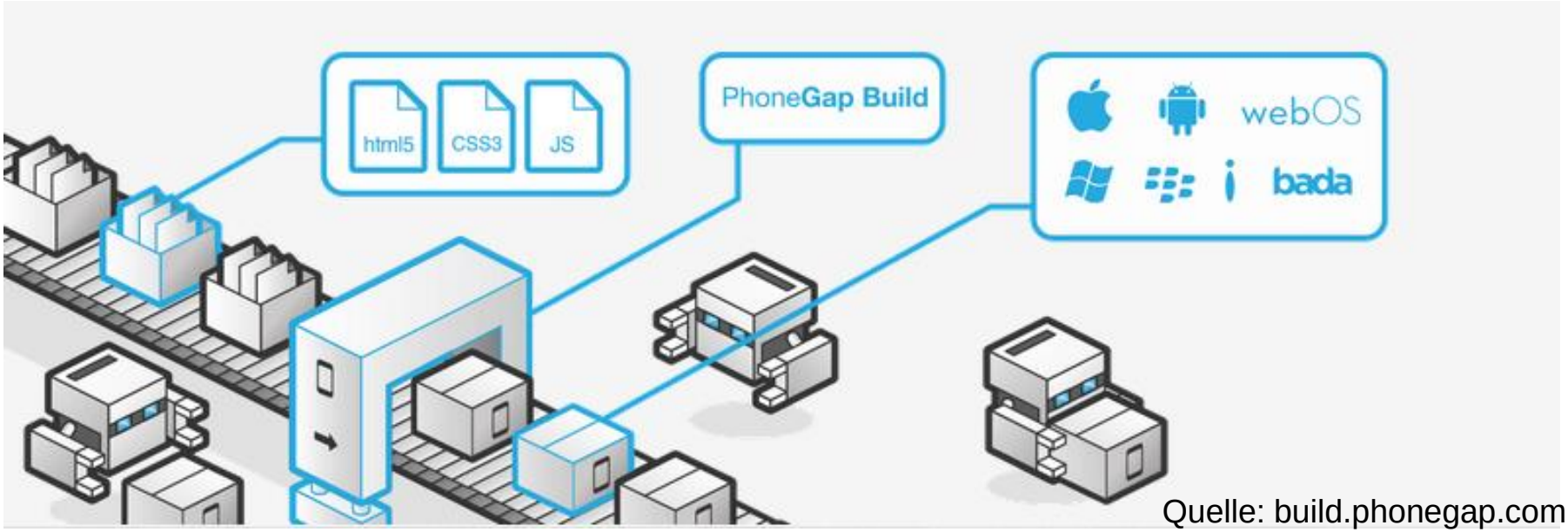
# PhoneGap

- Unterstützung von iOS, Android, BlackBerry, Windows Phone 7/8 sowie native Funktionalitäten (z.B. Kamerazugriff) für HP WebOS, Symbian und Bada OS
- Programmierung in reinem HTML5/JavaScript


- Ursprünglich von Nitobi Software entwickelt
- In 2011 von Adobe gekauft
- Seit Oktober 2011 ein Apache Projekt – „Apache Callback“ bzw. inzwischen „Apache Cordova“ – unter Apache Lizenz 2.0
- Nutzung der IDEs zur nativen Entwicklung
- Das Einbinden des geschriebenen Codes in eine Anwendung wird durch das Erstellen einer Web-View in nativem Code gelöst, der dann die HTML-Startseite der eigentlichen Anwendung aufruft

# PhoneGap Build

- Beim Kompilieren wird eine Library hinzugebunden, so dass die JavaScript-Befehle auch in native Befehle umgesetzt werden können
  - → Die kompilierte Anwendung enthält weiterhin den HTML-, CSS- und JavaScript-Code, der bei der Ausführung durch die integrierte Web-View interpretiert wird
  - → Performance geringer, da das Laden des Browsers in der Web-View länger dauert als das Laden einer nativen



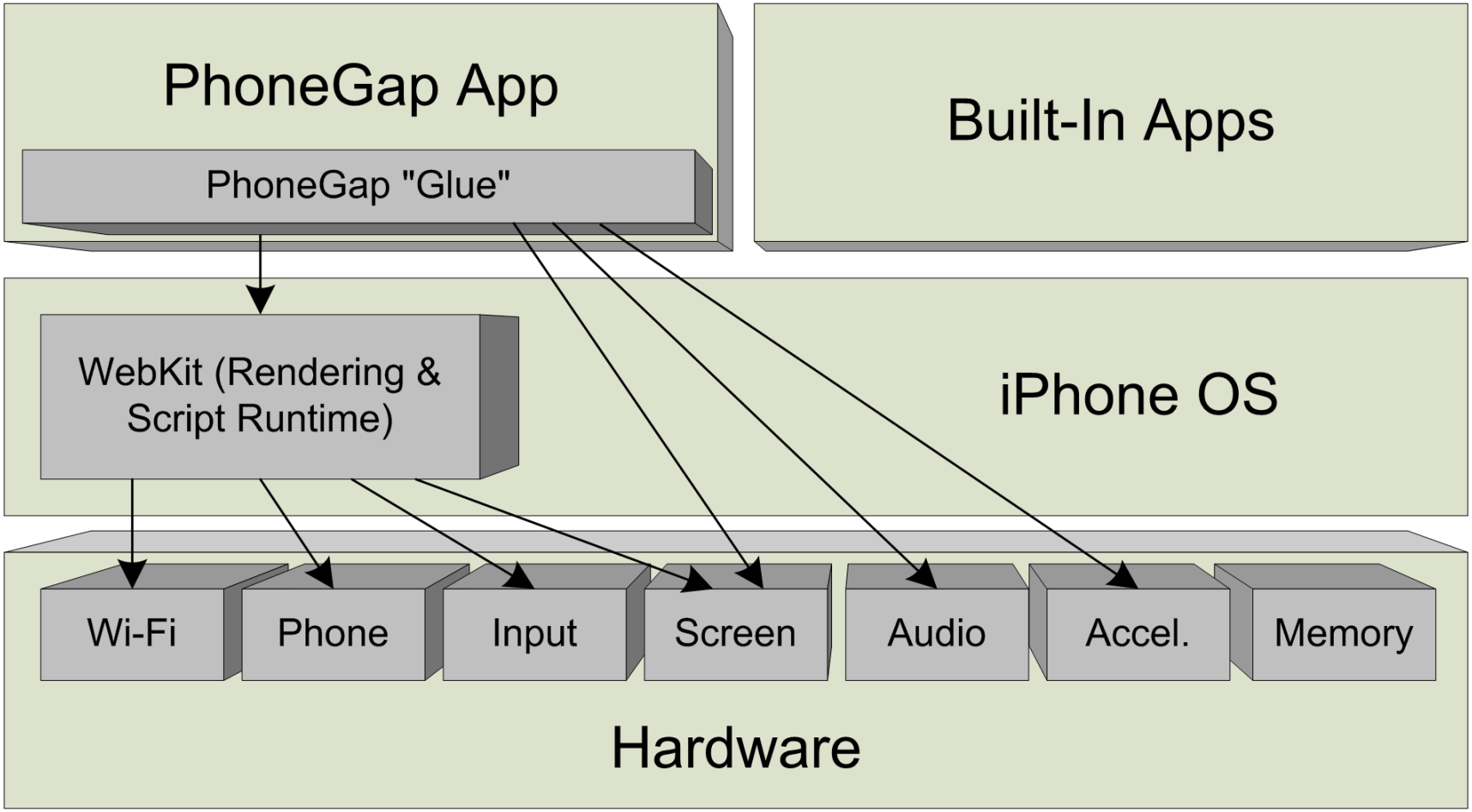

Quelle: build.phonegap.com

# PhoneGap

- Resultierende Anwendung nichts anderes als eine Webanwendung, daher ist das Debuggen und Testen mit herkömmlichen Werkzeugen der Webentwicklung möglich
  - Live-Debugging auf dem mobilen Endgerät z.B. mit *Weinre* möglich
- Für Oberflächengestaltung wird weiteres Framework benötigt,
  - Funktioniert gut z.B. mit jQuery Mobile, jQTouch und Sencha Touch

- Ein kostenpflichtiger Service für das automatisierte Build einer PhoneGap-Anwendung für alle Plattformen ist mittlerweile verfügbar, falls keine Build-Infrastruktur vorhanden ist.

# PhoneGap on iPhone



# Sencha Touch

- HTML5- bzw. JavaScript-Framework für mobile Endgeräte
- Touchbedienung optimiert
- Kann Oberflächen sehr plattformnah darstellen, z.B. im Vergleich zu jQuery mobile
- Unterstützung von iOS, Android, BlackBerry, Kindle Fire

# Sencha Touch


- Unterschiedliche Lizenzmodelle:
  - OpenSource GNU GPL license v3
  - Freie kommerzielle Nutzung auf Named Licence beruhend
  - Kostenpflichtige Lizenz bei Integration in kommerzielles SDK
- Programmierung überwiegend in JavaScript
  - Basiert auf Technologien des Ext JS-Frameworks, ein JS-Framework für Rich Internet Applications
  - Nur sehr wenige Stellen, z.B. die Art der Anzeige einzelner Elemente einer Liste, an denen man auf HTML-Elemente zurückgreifen muss
- Unterstützt ein Model-View-Controller-Pattern (MVC)

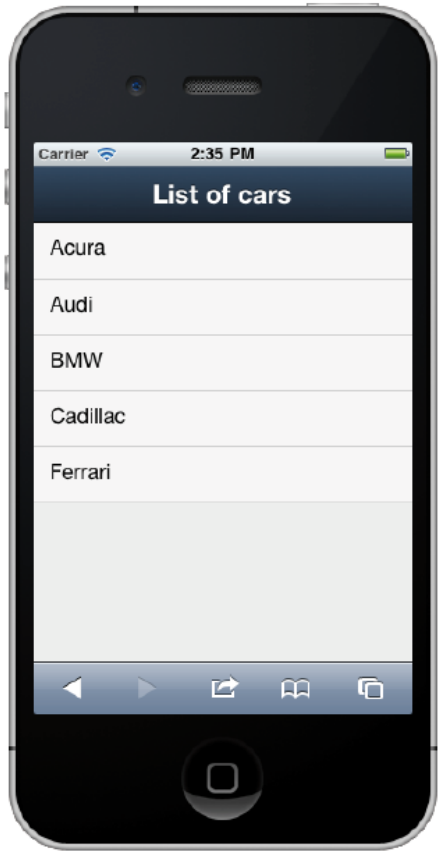
# Sencha Touch Beispiel

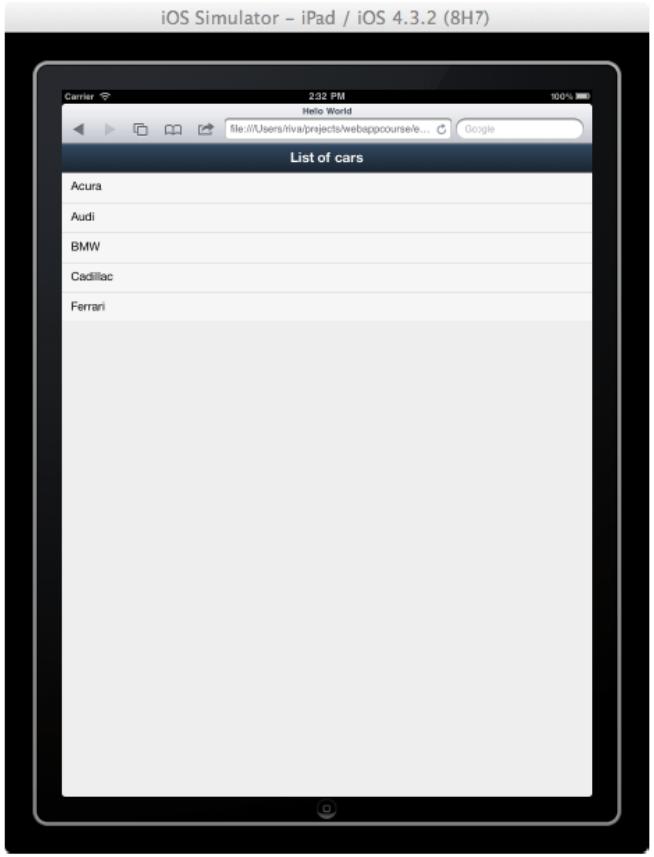
```
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
        <title>Hello World</title>
        <script src="sencha-touch.js" type="text/javascript"></script>
        <link href="sencha-touch.css" rel="stylesheet" type="text/css" />
         <script type="text/javascript">
            new Ext.Application({
                launch: function() {
                    var cars = new Ext.data.Store({
                        model: Ext.regModel('', {fields: ['name', 'link']}),
                        data: [
                            {name: 'Acura', link:'na'},
                            {name: 'Audi', link:'sa'},
                            {name: 'BMW', link:'eu'},
                            {name: 'Cadillac', link:'eu'},
                            {name: 'Ferrari', link:'eu'}
                        ]
                    });
                    new Ext.Panel({
                        fullscreen: true,
                        dockedItems: [{ xtype: 'toolbar', title: 'List of cars', }],
                        items: [{ xtype: 'list', store: cars, itemTpl: '{name}' }]
                    });
                }
            });
        </script>
    </head>
    <body></body>
</html>
```

Quelle: Claudio Riva 2012

# Sencha Touch Beispiel

## senchaSimple on a iPhone and iPad




Quelle: Claudio Riva 2012

# jQuery mobile

- Mobile und auf Gestensteuerung optimierte Version der JavaScript-Bibliothek jQuery
- OpenSource MIT Lizenz
- Unterstützung sämtlicher Plattformen

# jQuery mobile


- Entwicklung vollständiger HTML-Seiten
  - Einzelne Seiten werden mit Hilfe von -Elementen in Blöcke unterteilt, die über CSS gestaltet werden können
  - Die einzelnen -Elemente werden mit dem Attribut data-role versehen, welches die Werte page, header, content und footer beinhalten kann
  - Mehrere logische Seiten können über die Zuweisung einer id bei div-Blöcken mit der data-role page in einer physischen Seite dargestellt werden
  - Seitenübergänge können über data-transition angegeben werden, Dialoge über data-rel.
  - Theming sehr einfach über CSS3 möglich
- Keine Umsetzung des MVC-Patterns

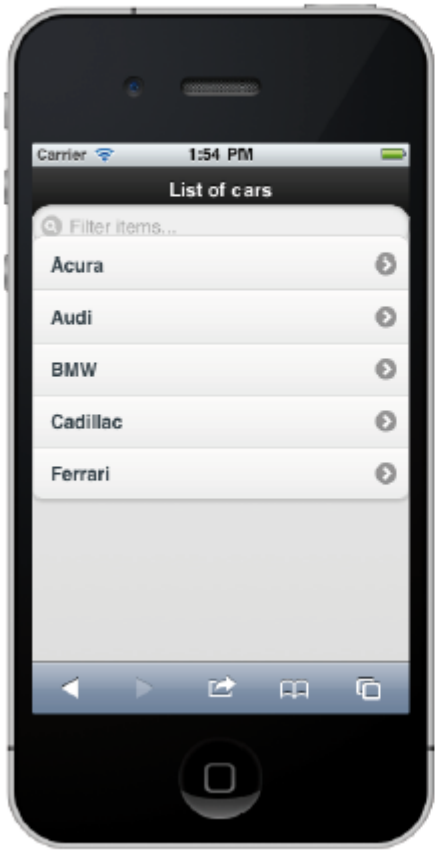
# jQuery mobile Beispiel

```
<!DOCTYPE html>
<html>
  <head>
  <title>Hello World</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="http://code.jquery.com/mobile/1.0/jquery.mobile-1.0.min.css" />
  <script type="text/javascript" src="http://code.jquery.com/jquery-1.6.4.min.js"></script>
  <script type="text/javascript" src="http://code.jquery.com/mobile/1.0/jquery.mobile-1.0.min.js"></script>
</head>
<body>
  <div data-role="page">
    <div data-role="header">
      <h1>List of cars</h1>
    </div><!-- /header -->
    <ul data-role="listview" data-inset="true" data-filter="true">
      <li><a href="#">Acura</a></li>
      <li><a href="#">Audi</a></li>
      <li><a href="#">BMW</a></li>
      <li><a href="#">Cadillac</a></li>
      <li><a href="#">Ferrari</a></li>
    </ul>
  </div><!-- /page -->
</body>
</html>
```

Quelle: Claudio Riva 2012

# jQuery mobile Beispiel

## jQuerySimple on a iPhone and iPad





Quelle: Claudio Riva 2012

# OFFLINE MODUS

# Offline Nutzung - Lokales Speichern

Für die Offline Nutzung einer Mobile Web App ist folgendes zu beachten:

- Lokales Speichern der benötigten Daten über localStorage
- Definition der lokal zu cachenden Dateien über ein manifest File
- Managen der Verbindungsänderungen mit online und offline Events
- Definition einer Synchronisationsstrategie

**4 Möglichkeiten** zur lokalen Speicherung von Daten:

- Web Storage mit localStorage und sessionStorage
- WebSQL
- IndexedDB
- Dateisystemzugriff

# Web Storage

- HTML5 führt zwei Key-Value Speichermechanismen ein:
- **localStorage**: persistent über Browser Neustarts
- **sessionStorage**: Daten werden bei Ende der Browser Session gelöscht

```
//set a key
localStorage.myData = "Hello World";
localStorage.setItem('myData', 'Hello World');
//get a key
var data;
data = localStorage.myData;
data = localStorage.getItem('myData');
//get key name at specified position
var name;
name = localStorage.key(0)
//delete a key
localStorage.myData = undefined;
localStorage.removeItem('myData');
//remove all key value pairs
localStorage.clear();
```

# Web Storage

- Es sind nur String-to-String Mappings möglich (UTF-16)
  - → Es können nur Strings gespeichert werden
  - → Keys können nur Strings sein
- JSON Serialisierung für das Speichern komplexerer Datenstrukturen
- Es gilt „same origin policy“
  - hostname, port and protocol
- Ca. 2-5 MB Speicherung pro Host
  - Vgl. Cookies: max. 20 Cookies à 4KB
- Performance besser als bei Cookies aber nicht die schnellste (abhängig vom Browser)

# WebSQL

- Bietet strukturierte relationale SQL DB
- Aufbau eines DB Schemas nötig
- Danach Ausführen klassischer SQL Anfragen möglich
  - Z.B. dann Unterstützung von Tabellen: create, insert, update etc.

- Transaktional
- Über Browser Sessions persistent
- Arbeit des W3C wurde im November 2010 eingestellt!

# IndexedDB

- Versucht WebStorage und WebSQL zu kombinieren
- Speichermechanismus für unstrukturierte Daten (NoSQL)
- Daten können als **Key-Value-Pairs** gespeichert werden, wobei Values komplexe Datenstrukturobjekte sein können
- Es können mehrere DBs definiert werden
- Keine vorherige Schemadefinition nötig
- Auf einem transaktionalen Model basierend
- Gute Performanz
  - Daten werden indexiert
  - Asynchrone API → UI Thread wird nicht blockiert

# Dateisystemzugriff

- Erlaubt Lesen, Schreiben und Navigieren der Dateisystemhierarchien
- Elementar für das Managen und Speichern großer Dateien oder Binärinhalten auf Client-Seite → unlimitiertes Speichern
- Asynchrone API → blockiert nicht die UI der App
- Nicht transaktional
- Lookup-Funktionen müssen selbst implementiert werden

- Vorgehen:
  - Persistenten oder temporären Dateisystemzugriff anfragen
  - Dann sind CRUD Operationen auf Dateien und Ordner möglich
- Persistent vs. Temporär:
  - Temporär gespeicherte Daten der App können vom Browser gelöscht werden
  - Persistente Daten können vom Browser nicht ohne Freigabe durch die App gelöscht werden

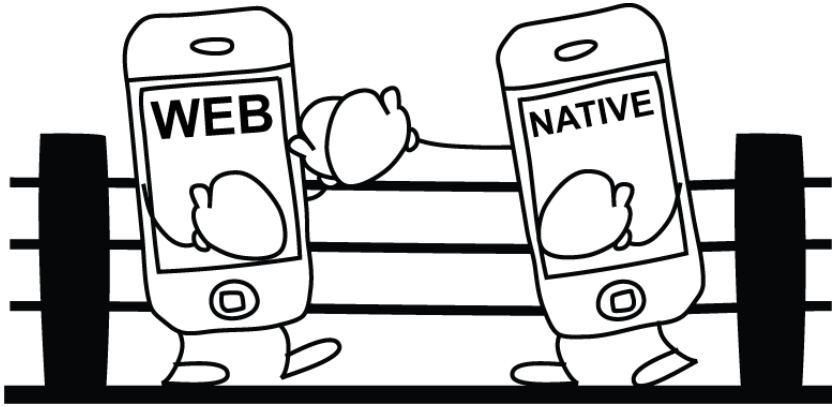
# Dateisystemzugriff

```
window.requestFileSystem(type, size, successCb, [errorCb])
```

- type
  - LocalFileSystem.TEMPORARY
  - LocalFileSystem.PERSISTENT
- size
  - benötigte Speichergröße in Bytes
- successCb
  - Success Callback, Argument ist ein FileSystem Objekt
- errorCb
  - Error Callback, Argument ist ein FileError Objekt

# Native vs. Web App

- Zu berücksichtigende Faktoren:
  - Benutzer Oberfläche (look & feel)
  - Entwicklung (Plattform, Sprache…)
  - Fähigkeiten (Gerät eigene Eigenschaften)
  - Monetarisierung (Zahlung, Werbung…)
  - Übertragunsverfahren (Download, Installieren, Updates…)
  - Versionierung der App (Nutzer Updates vs. Web Update)
  - Stärke vs. Schwäche (Performance, Aufwand…)



# Native App vs. Mobile Web App


Quelle: mobilemetrics.de

# MOBILE WEB APPS
# MIT GOOGLE WEB TOOLKIT

# GWT - Google Web Toolkit

- Toolset für komplexe AJAX Web Anwendungen
- OpenSource Lizenz: Apache Lizenz 2.0
- Unterstützung von Windows, Linux, Mac OS
- Entwicklung vollständiger Web-Seiten mit **Java**
  - Java Code wird in XHTML und JavaScript übersetzt
  - Code ist optimiert
  - Theming sehr einfach über CSS möglich
- MVP Pattern (MVC-Pattern auch möglich)
- Plugins für IDEs z.B. Eclipse

# GWT - Vorteile

- Dynamische und wiederverwendbare UI Komponenten
- Einfaches RPC-Verfahren
- Browser History Management
- Unterstützung für Java Debugging
- Cross-browser
- Internationalisierung und Lokalisierung
- HTML5 Canvas
- JavaScript Native Interface (JSNI): Integration von existierendem JavaScript in den Java Source Code
- Unterstützung für Google APIs in GWT Apps
- Open-source
- Zahlreiche Libraries (von Google und dritten Parteien)

# GWT – Die Toolbox

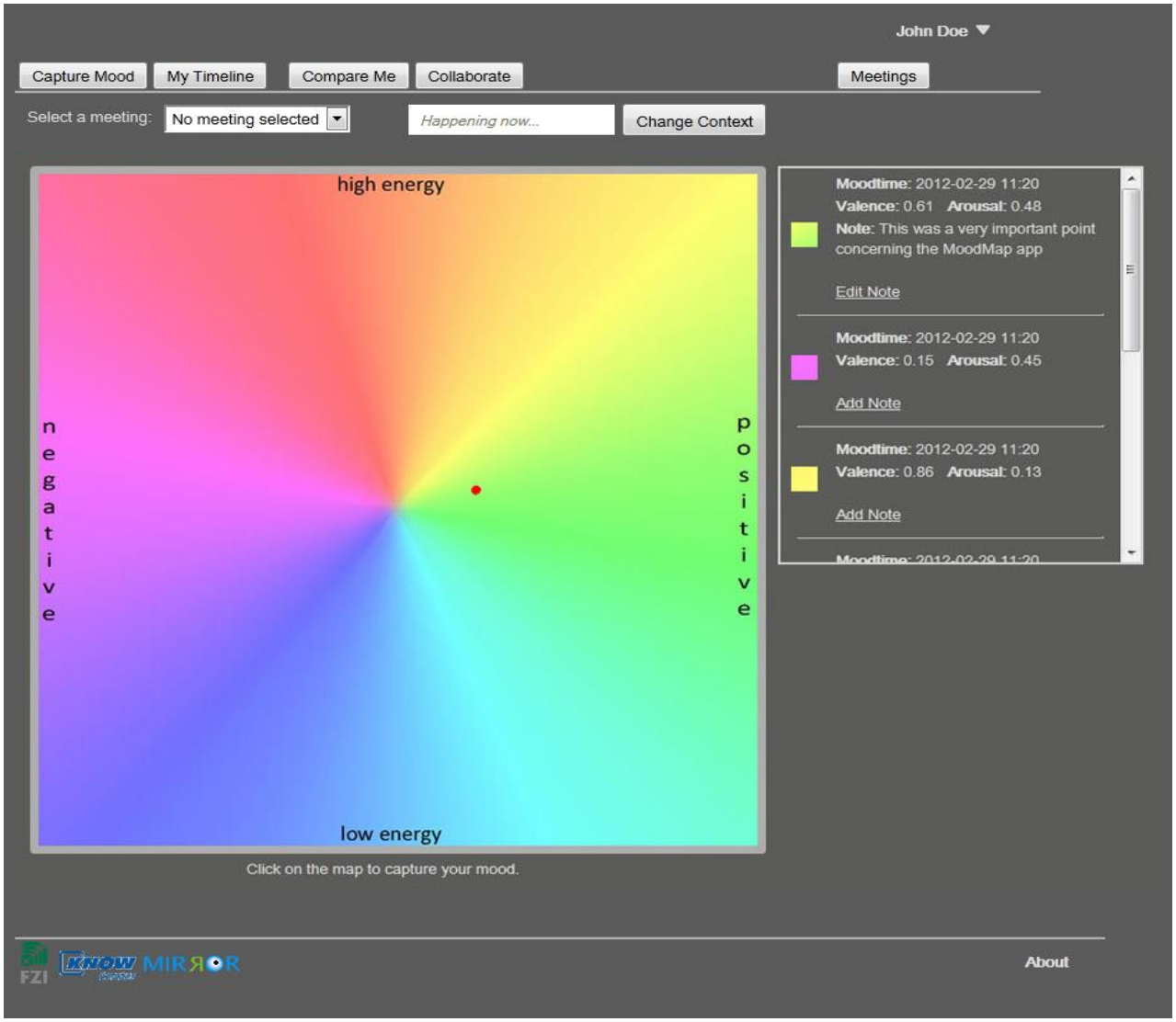
- SDK
  - Entwicklung
  - Java API Bibliotheken, Compiler und Entwicklungsserver

- Speed Tracer
  - Google Chrome Extension - Leistungsprobleme identifizieren

- Plugin für Eclipse
  - Google Web Toolkit + Google App Engine

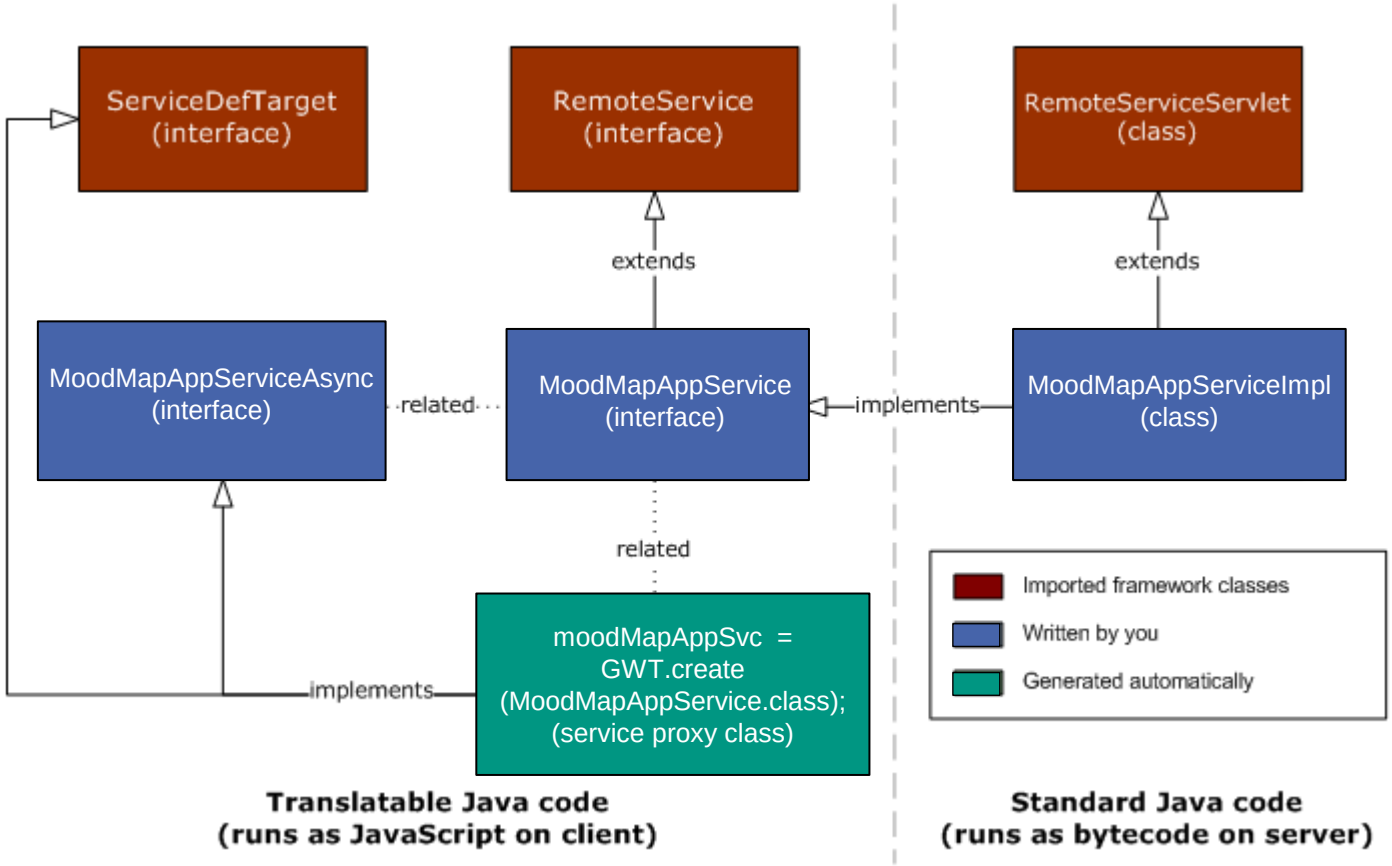
- GWT Designer
  - Gestaltung von User Interfaces

# Das MoodMap App Beispiel – UI



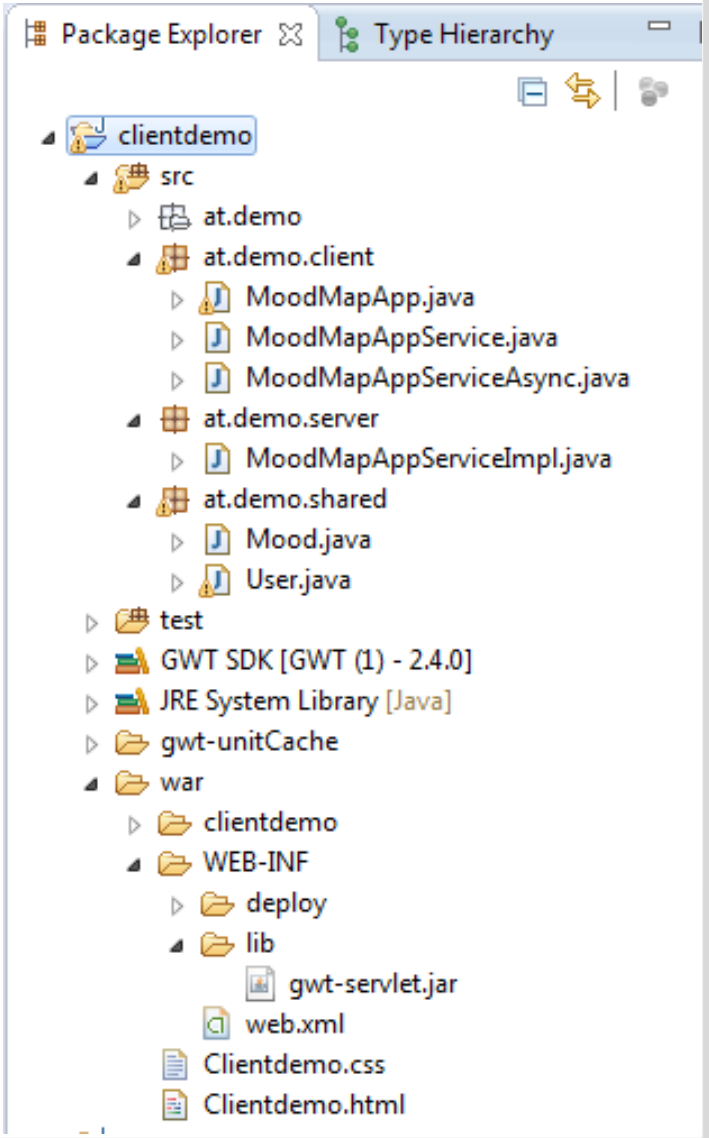
# Das MoodMap App Beispiel – RPC

- Remote Procedure Calling
  - Client-Server Kommunikation ohne expliziten Code



# Das MoodMap App Beispiel - RPC

- Client Side
  - ***MoodMapApp.java:*** Klasse mit Benutzeroberfläche und Kommunikation mit Server
  - ***MoodMapAppService.java*** : Interface für den Service, mit allen RPC Methoden.
  - ***MoodMapAppServiceAsync.java*** : Asynchrones Interface für den Service, der von Client Seite aufgerufen wird.
- Server Side
  - ***MoodMapAppServiceImpl.java***: Klasse, die RemoteServiceServlet erweitert und die Implementation von dem Service enthält.



# Das MoodMap App Beispiel – Code Client

- ***MoodMapApp.java:*** EntryPoint, UI Komponenten und Aufrufe

```
package at.demo.client;

import at.demo.shared.Mood;

 * Entry point classes define <code>onModuleLoad()</code>.
public class MoodMapApp implements EntryPoint {

    /** Create a remote service proxy to talk to the server-side   */
    private final MoodMapAppServiceAsync mmas = GWT.create(MoodMapAppService.class);

    /** This is the entry point method.  */
    public void onModuleLoad() {

        /** Declaration if UI Components  */
        final Image                 moodBackgroundImage  = new Image();
        final Image                 myMoodImage          = new Image();
        final AbsolutePanel         drawArea             = new AbsolutePanel();
        final Button                usersButton          = new Button("Get users");

        /** Construction of UI  */
        myMoodImage.setUrl("images/mymood.gif");
        moodBackgroundImage.setSize("600", "600");
        moodBackgroundImage.setUrl("images/moodmap.png");

        usersButton.setStyleName("buttonstyle");

        drawArea.add(myMoodImage);
        drawArea.add(moodBackgroundImage);

        RootPanel.get("moodmap").add(drawArea);
        RootPanel.get("moodmap").add(usersButton);

        /** Interaction with UI  */
        moodBackgroundImage.addMouseDownHandler(new MouseDownHandler() {
            public void onMouseDown(MouseDownEvent event) {

                //Show red point
                int moodImageOffset = 5;
                drawArea.add(myMoodImage, event.getX() - moodImageOffset , event.getY() - moodImageOffset);

                //Create and save mood
                Mood m = new Mood(getCurrentTime(), event.getX(), event.getY());
                saveMyMood(m);

            }
        });
    }
```

```
package at.demo.client;

import at.demo.shared.Mood;

 * Entry point classes define <code>onModuleLoad()</code>.
public class MoodMapApp implements EntryPoint {

    /** Create a remote service proxy to talk to the server-side   */
    private final MoodMapAppServiceAsync mmas = GWT.create(MoodMapAppService.class);

    /** This is the entry point method.  */
    public void onModuleLoad() {

        /** Declaration if UI Components  */
        final Image                         moodBackgroundImage  = new Image();
        final Image                         myMoodImage          = new Image();
        final AbsolutePanel                 drawArea             = new AbsolutePanel();
        final Button                        usersButton          = new Button("Get users");

        /** Construction of UI  */
        myMoodImage.setUrl("images/mymood.gif");
        moodBackgroundImage.setSize("600", "600");
        moodBackgroundImage.setUrl("images/moodmap.png");

        usersButton.setStyleName("buttonstyle");

        drawArea.add(myMoodImage);
        drawArea.add(moodBackgroundImage);

        RootPanel.get("moodmap").add(drawArea);
        RootPanel.get("moodmap").add(usersButton);

        /** Interaction with UI  */
        moodBackgroundImage.addMouseDownHandler(new MouseDownHandler() {
            public void onMouseDown(MouseDownEvent event) {

                //Show red point
                int moodImageOffset = 5;
                drawArea.add(myMoodImage, event.getX() - moodImageOffset , event.getY() - moodImageOffset);

                //Create and save mood
                Mood m = new Mood(getCurrentTime(), event.getX(), event.getY());
                saveMyMood(m);

            }
        });
    }
```

# Das MoodMap App Beispiel – Code Client

- ***MoodMapApp.java:*** EntryPoint, UI Komponenten und Aufrufe

```
private void saveMyMood(Mood m){
    // Set up the callback object.
    AsyncCallback<Void> callback = new AsyncCallback<Void>() {

        public void onFailure(Throwable caught) {
            // Do something with errors.
            Window.alert("User addition failed, reason: " +caught.getMessage());
        }

        public void onSuccess(Void result) {
            // Do something with the results
            Window.alert("Mood saved successfully ");
        }
    };

    //Make the call to the mood service.
    mmas.saveMood(m, callback);

}
```

# Das MoodMap App Beispiel – Code Client

- ***MoodMapAppService.java*** : Interface für den Service, mit allen RPC Methoden.

```
package at.demo.client;

import java.util.List;

import com.google.gwt.user.client.rpc.RemoteService;
import at.demo.shared.Mood;
import at.demo.shared.User;


/**
 * Synchronous interface - implemented by the server
 * The client side stub for the RPC service.
 */

public interface MoodMapAppService extends RemoteService {

    List<User> getUsers() throws IllegalArgumentException;

    void saveMood(Mood m) throws IllegalArgumentException;

}
```

# Das MoodMap App Beispiel – Code Client

- ***MoodMapAppServiceAsync.java*** : Asynchrones Interface für den Service, der von Client Seite aufgerufen wird.

```
package at.demo.client;

import java.util.List;

import at.demo.shared.Mood;
import at.demo.shared.User;

import com.google.gwt.user.client.rpc.AsyncCallback;


/**
 *  Asynchronous interface, which is called by the client-side
 */
public interface MoodMapAppServiceAsync {

    void getUsers(AsyncCallback<List<User>> callback);

    void saveMood(Mood m, AsyncCallback<Void> callback);

}
```

# The MoodMap App Beispiel – Code Server

- ***MoodMapAppServiceImpl.java***: Implementation des Services auf der Server Seite.

```java
package at.demo.server;

import java.util.ArrayList;
import java.util.List;


import at.demo.client.MoodMapAppService;
import com.google.gwt.user.server.rpc.RemoteServiceServlet;

import at.demo.shared.Mood;
import at.demo.shared.User;

/**
 *  Implementation of the service (server)
 */
public class MoodMapAppServiceImpl extends RemoteServiceServlet implements MoodMapAppService{

    private static final long serialVersionUID = -4537317369125726854L;

    /**
     * Get list of users
     * @return List of users
     */
    public List<User> getUsers() throws IllegalArgumentException {
        List<User> res = new ArrayList<User>();

        //Retrieve List of users from the Database in variable res

        return res;
    }


    /**
     * Save a mood in the Database
     * @return Void
     */
    public void saveMood(Mood m) throws IllegalArgumentException {

        //Save mood in the database

    }
}
```

```java
package at.demo.server;

import java.util.ArrayList;
import java.util.List;


import at.demo.client.MoodMapAppService;
import com.google.gwt.user.server.rpc.RemoteServiceServlet;

import at.demo.shared.Mood;
import at.demo.shared.User;

/**
 *  Implementation of the service (server)
 */
public class MoodMapAppServiceImpl extends RemoteServiceServlet implements MoodMapAppService{

    private static final long serialVersionUID = -4537317369125726854L;

    /**
     * Get list of users
     * @return List of users
     */
    public List<User> getUsers() throws IllegalArgumentException {
        List<User> res = new ArrayList<User>();

        //Retrieve List of users from the Database in variable res

        return res;
    }


    /**
     * Save a mood in the Database
     * @return Void
     */
    public void saveMood(Mood m) throws IllegalArgumentException {

        //Save mood in the database

    }
}
```

# Google Web Toolkit Bewertung


(+) Trennung von Anwendung und Darstellung

(+) Einfach erlernbar und schnell anwendbar – kein JavaScript

(+) Stetig wachsende Community – guter Support und Dokumentation

(+) Gewohnte IDE kann genutzt werden

(+) Entwicklung der GUI entspricht grundlegend der Entwicklung durch AWT bzw. Swing

(+) Unit-Tests sind möglich

(+) Anwendung kann im Hosted- sowie Web-Mode getestet werden

(+) Durch RPC-Schnittstelle muss sich der Entwickler nicht mehr direkt um die Details der asynchronen Kommunikation mittels *HttpXMLRequest* oder der Serialisierung von Objekten kümmern

# Google Web Toolkit Bewertung


(-) Getrennte Betrachtung von Client und Serverseite (Bereitstellung von Daten durch SOAP basierte RPC Calls)

(-) Proprietärer Compiler

(-) Generiertes JavaScript ist kryptisch (proprietär), Debugging nur in Java

(-) JavaScript-Compiler und *Hosted Mode Browser* sind nicht Open-Source und dürfen nicht weitergegeben werden bzw. nur mit Genehmigung von Google

(-) Google behält sich vor, die Lizenzbedingungen in der Zukunft jederzeit zu ändern

# Was haben wir gelernt?

- Generelle Kenntnis der Historie von mobilen Technologien und verfügbaren Betriebssysteme
- Technologien zur Entwicklung von mobilen Anwendungen auswählen und anwenden können
- Anforderungen zur Erstellung von native- und Cross-Plattform-Entwicklung von  mobilen Anwendungen
- Kenntnisse für die Erstellung eine eigene Web-Anwendung mit Google Web Toolkit

# Literatur


- Marc Pilgrim *HTML5: Up and Running* O'Reilly Media 2010
- Maximiliano Firtman *Programming the Mobile Web* O'Reilly Media 2010
- Claudio Riva: *Mobile Web Appplications Development with HTML5* – Lecture 1: Introduction, Aalto University Fall 2012, http://aaltowebapps.com/docs/2012_2/lecture1.pdf
- Joshua Marinacci *Building Mobile Applications with Java.* O'Reilly Media 2012
- Heidenreich, M. & Kockert, T.: Vervielfältigungswerk - Drei Frameworks für mobile Apps im Vergleich. iX Magazin für professionelle Informationstechnik, Februar 2012
- Google Developers – Google Web Toolkit https://developers.google.com/web-toolkit/
- GWT Literature suggestions - http://www.gwtproject.org/books.html
- GWT - Communicate with a Server
  https://developers.google.com/web-toolkit/doc/latest/DevGuideServerCommunication?hl=en
- Android Developer - http://developer.android.com/guide/components/index.html