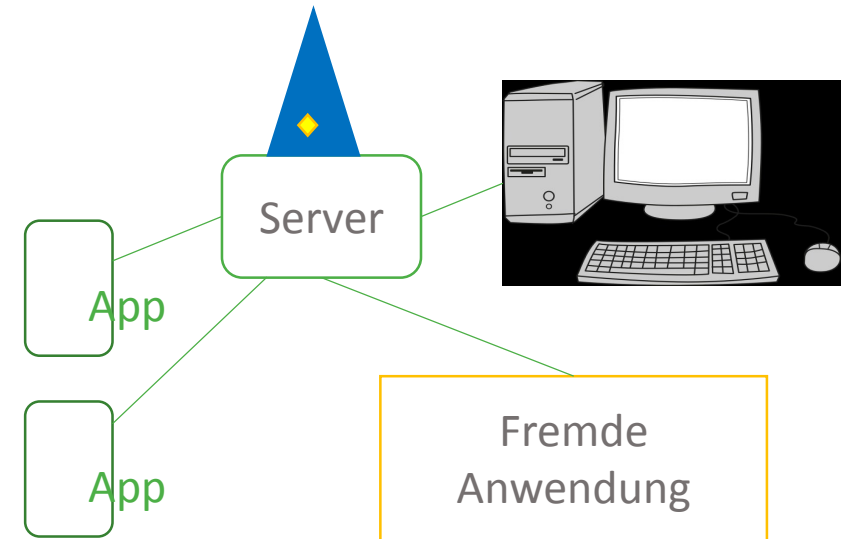


# Eine Architektur für die Entwicklung von Umweltinformationssystemen auf Basis von mit Android-Apps aufgenommenen Daten

*Christine Müller*



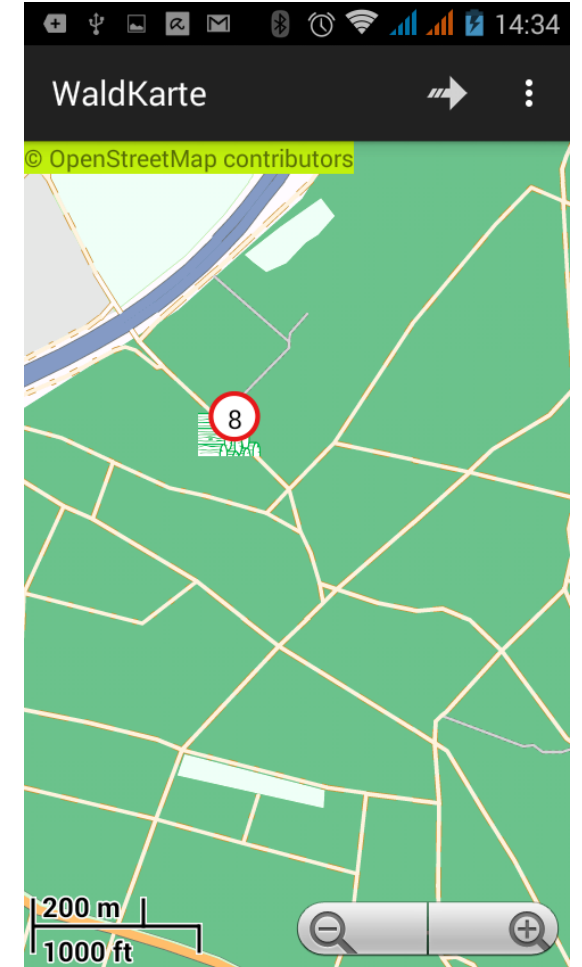
# Apps in der Forstwirtschaft

- **WaldFliege**: App zur Aufnahme von Holzpoltern im Wald.
- **WaldKarte**: App zur Darstellung von Offline-Karten mit der Möglichkeit, Lagerplätze anzeigen zu lassen.

The screenshot shows the 'Einzelstamm' (Single Tree) app interface. It features several input fields and buttons for recording tree data:

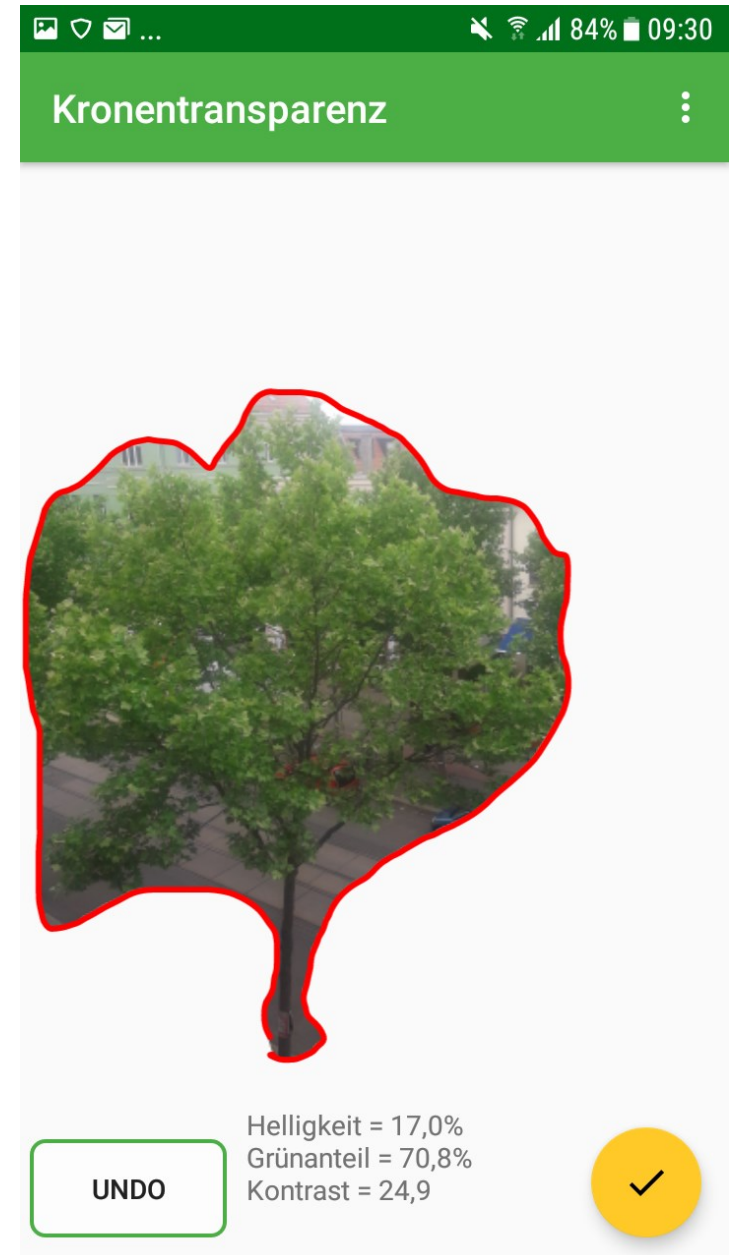
- Hiebnr./Losnr.:** A dropdown menu with '001/1002' selected.
- Polternr.:** A dropdown menu with '3' selected.
- Waldort:** A text field containing 'kein Waldort' (no forest location).
- Stammnummer:** A text field with '28' and a button labeled '123'.
- Letzter:** A text field with '0,39 fm'.
- Polter:** A text field with '11,72 fm' and a button labeled 'zur Stammliste' (to tree list).
- Baumart:** A dropdown menu with 'Fichte' (Spruce) selected.
- Güte:** A dropdown menu with 'B' selected.
- Länge [m]:** A text field with a green underline.
- Durchmesser [cm]:** A text field with 'KI' entered.

Buttons for 'Länge gleich' (Length same) and 'KI' are also visible.



# Apps in der Forstwirtschaft

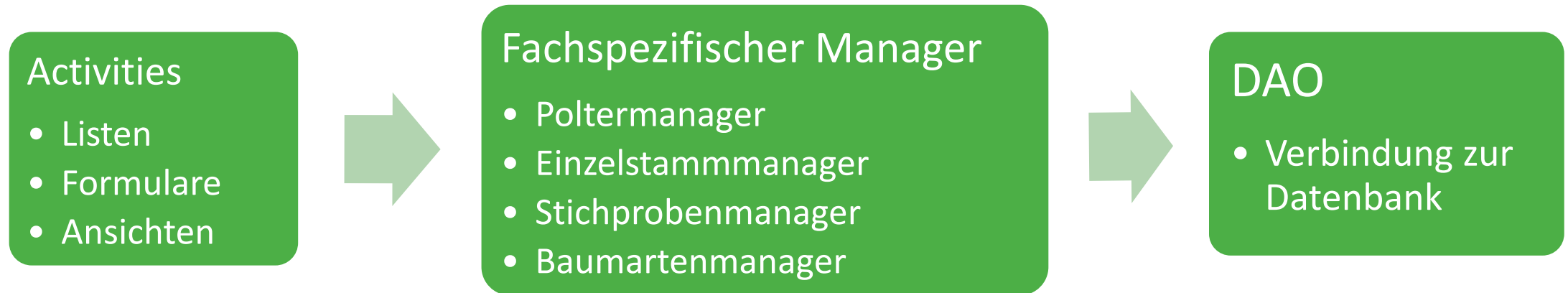
- Umwelt-Monitoring
- **Kronentransparenz**: App zur optischen Analyse von Baumkronen



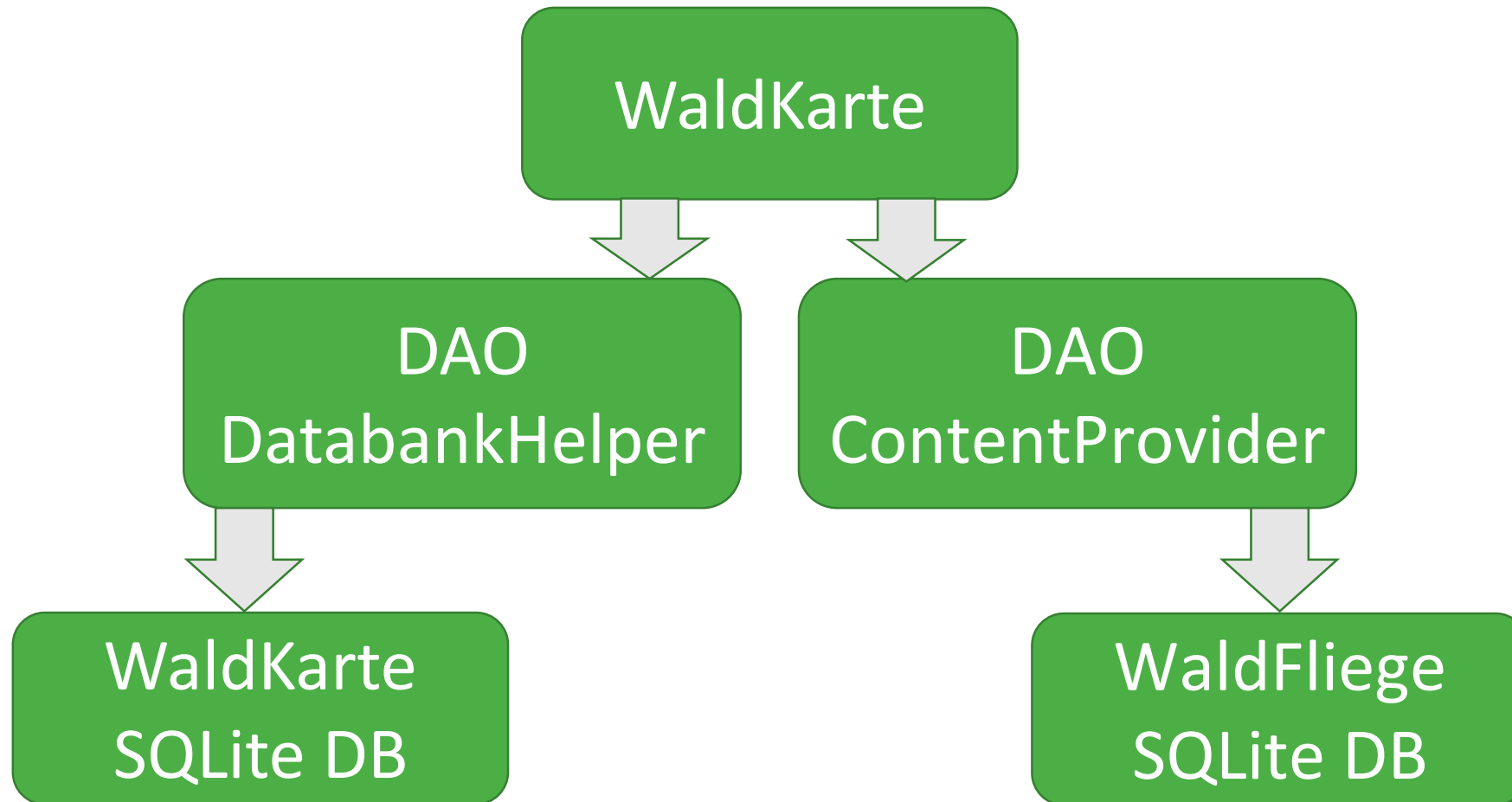
# Der Weg des Polters



# Model-View-Controller innerhalb der App

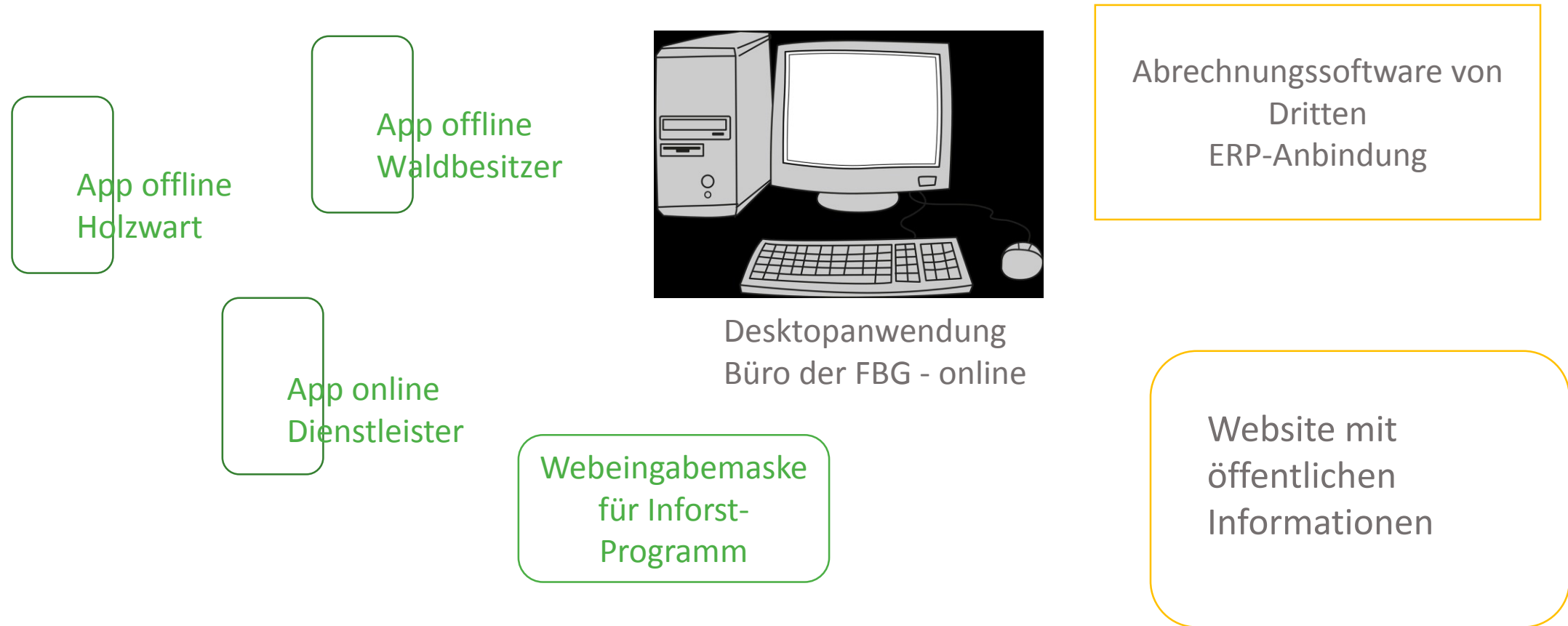


# Kommunikation zwischen Apps



# Herausforderungen für Schnittstellen

- Oder: Von der Einzelanwendung zum Umweltinformationssystem



# Fragen für die Schnittstellengestaltung

- Wie können diese Daten in einer zentralen Datenbank gespeichert und über eine Webanwendung den berechtigten Nutzern richtig angezeigt werden?
- Welche Mechanismen für die Synchronisation der Daten sind sinnvoll?
  - Offline / Online – Synchronisation: Ein Nutzer kann offline aktuellere Informationen haben, aber keiner weiß es
  - On Demand oder automatisch?
- Wie sollen Konflikte beim Einspielen in die Datenbank behandelt werden?
  - Kann man allgemeine Regeln festlegen?
  - Gibt es einen „allmächtigen“ Administrator?
  - Kann man Vertretungen temporär einrichten?



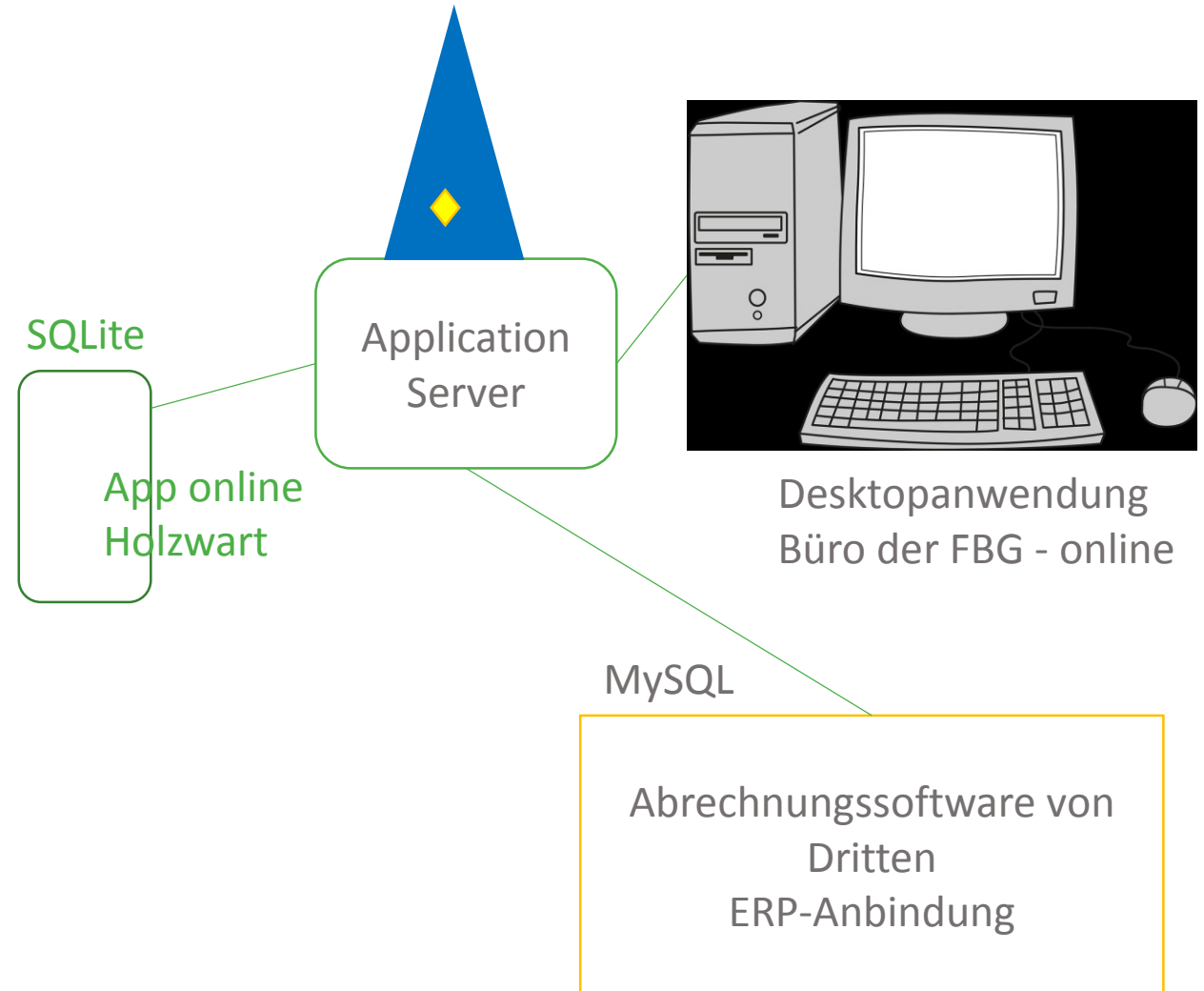
- Eigenschaften von REST:
  - Ressourcen mit eindeutiger Id = URI , z.B. :  
de.inforst.waldfliege/beispielkunde/Einzelstaemme/123  
de.inforst.waldfliege/beispielkunde/Aushaltung/Buche/Pollmeier
  - Verknüpfungen, Links
  - Standardmethoden:
    - GET, POST, PUT, DELETE, HEAD,
  - Unterschiedliche Repräsentationen
  - Statuslose Kommunikation

# Vorteile von REST

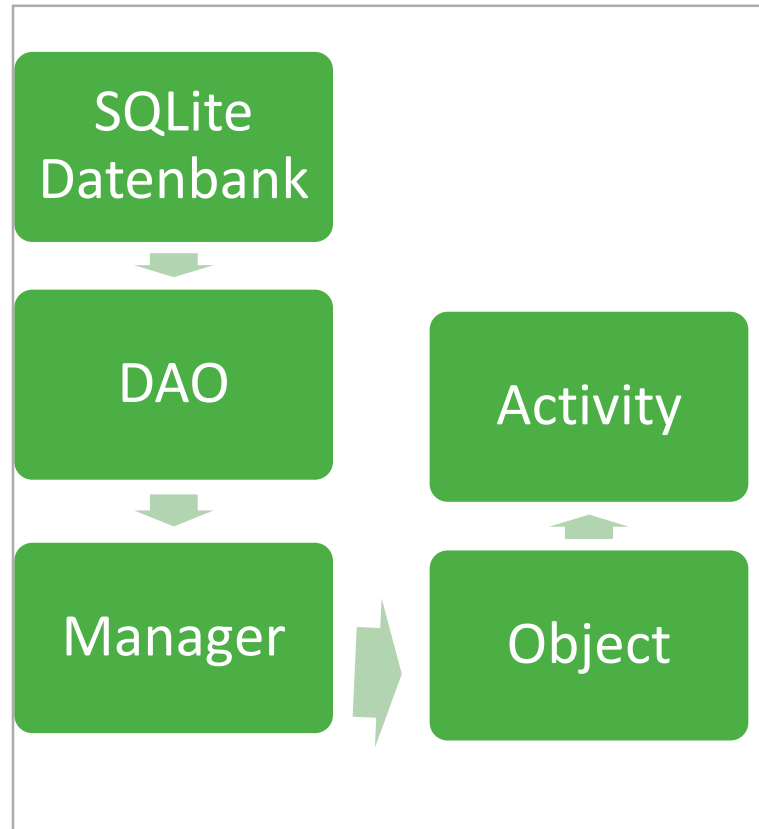
- Lose Kopplung – modularer Aufbau der Anwendung, Einbindung von Software anderer Anbieter
- Interoperabilität – Alle Komponenten können miteinander kommunizieren, Objekte sind anhand Ihrer URI von überall identifizierbar.
- Wiederverwendung – Abfragen, Views, Clientanwendungen etc. können wiederverwendet werden.
- Performance und Skalierbarkeit – aufgrund der möglichen Kleinteiligkeit der ausgetauschten Objekte gut skalierbar.

# Vorteile von REST

- Darstellbarkeit der Informationen in Apps, Web-Anwendungen und Desktop-Anwendungen
- Austausch der Informationen über verschiedene Datenbanken: SQLite, PostgreSQL, MySQL
- Unterschiedliche Repräsentationen derselben Objekte für verschiedene interne und externe Anwender

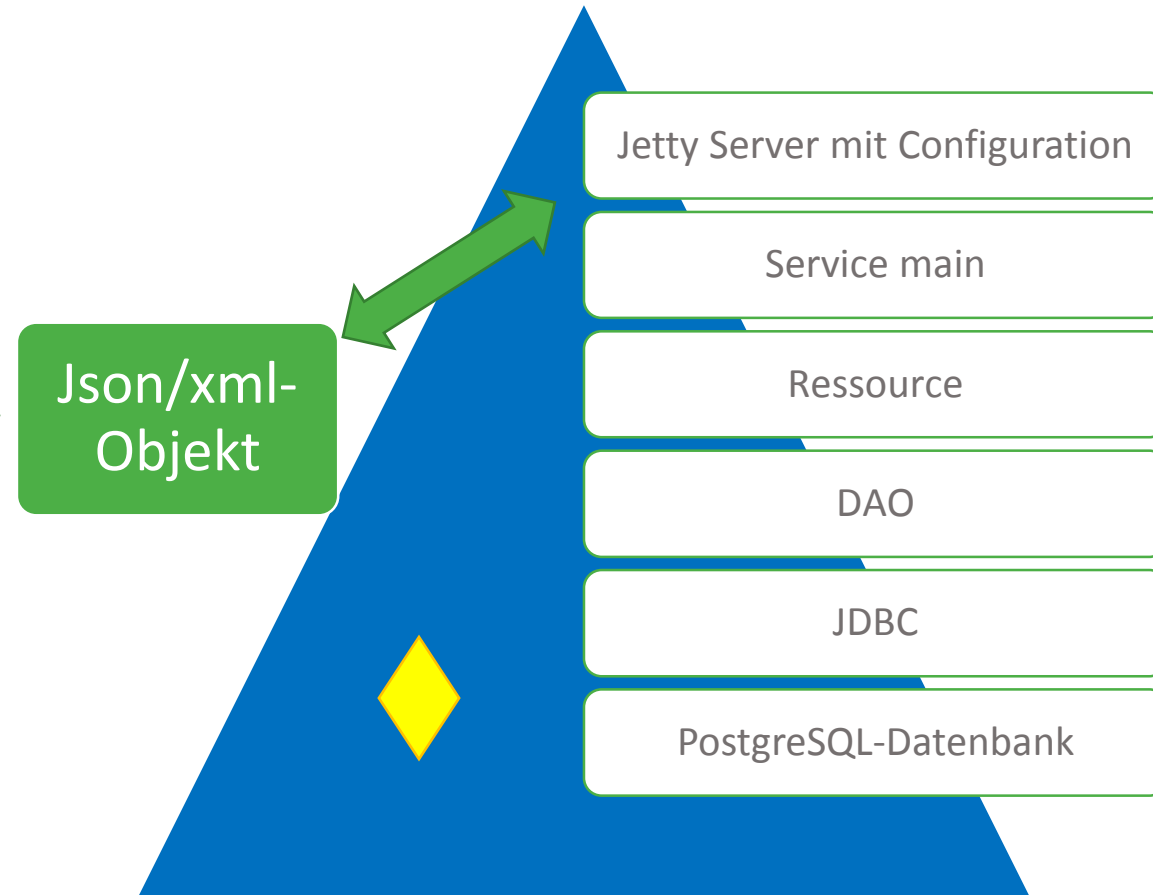


# REST-Service mit Dropwizard



Android - Client

08.06.2018



Webserver mit Dropwizard

UIS 2018 Nürnberg

# Dropwizard – mehr eine Sammlung als ein Framework

- Jetty für HTTP
- Jersey für REST JAX-RS
- Support für OAuth und HTTPS-Übertragung
- Metrics für Logging
- Hibernate für Mapping von Objekten in die Datenbank
- Jackson für Json
- JDBI
- Liquibase
- Freemaker / Mustache

# Vorteile von Dropwizard

- Flexibilität – einzelne Elemente können genutzt werden. Wenn man sie nicht nutzt dauert die Umsetzung länger, ist aber möglich.
- Gute Kombination bewährter Software-Tools
- Einfaches Deployment mit fat jar.
- Gute Performance
- Unterstützt Internet Sicherheit

- Authentizität – Ist es wirklich ein berechtigter App-Nutzer, der hier Daten schickt?
- Integrität – Daten werden nicht verändert während des Transports
  - Erreichbar durch SSL-Zertifikat und OAuth
- Vertraulichkeit – Können nur berechtigte Nutzer auf bestimmte Informationen zugreifen?
  - Erreichbar durch OAuth
- Verfügbarkeit - Kann der Server durch ständige Anfragen von Schadprogrammen lahmgelegt werden?
  - Zum Teil Erreichbar durch Zugangsbeschränkung für App, HealthChecks, Response-Regeln
- Verbindlichkeit – Kann man nachweisen, wer welche Aktionen auf dem Server durchgeführt hat?
  - Erreichbar durch Metrics

- Inhalte: Unerwünschte Bilder, unerwünschter Text. Bisher nur manuelle Kontrolle möglich, kein Problem bei Geschäftsanwendung, schon bei Endkunden-Anwendung.
- Offline/Online-Synchronisation: Nur möglich durch klare, systemübergreifende Geschäftsregeln. Automatische Synchronisation notwendig.
- Doppelte Geschäftslogik: Alle Geräte, die offline arbeitsfähig sein müssen, müssen die Geschäftslogik implementieren. (Lösung durch Regelmaschinen?)
- Doppelte Datenhaltung: Alle Geräte die offline arbeitsfähig sein müssen, müssen über eigene Datenbanken verfügen. Das führt zu doppelten Primärschlüsseln für die Objekte.



# Quellen und weiterführende Literatur

- <https://www.dropwizard.io/1.3.2/docs/>
- Starke, Gernot „Effektive Software Architekturen“, Carl Hanser Verlag, München 2011
- Tilkov, Stefan „Rest und HTTP“, dpunktVerlag GmbH, Heidelberg 2011