Architektur von Informationssystemen

Hochschule für angewandte Wissenschaften

Sommersemester 2018

Nils Löwe / nils@loewe.io / @NilsLoewe

4. Praktikum

Praktikum 4: Architekturbewertung

Fragen?

Wiederholung

Was ist Softwarearchitektur?

Geschichte und Trends

Sichten auf Architekturen

Qualiät und andere nichtfunktionale Anforderungen

Architekturmuster

Dokumentation von Architekturen

Technologien und Frameworks

Ruby on Rails

NodeJS

AngularJS

Microservices (Github)

AMQP

Twitter Bootstrap

Blockchain (Chainstep)

Ruby on Rails

Ruby on Rails

Ruby: Überblick

- Erste Version 1993
- Verbreitet seit 2006 (durch Rails)
- Objektorientiert
- Interpretiert
- Dynamisch getypt
- "Script Sprache"

Ruby on Rails

Ruby: Tradeoffs

- Flexibilität vs. Sicherheit
- Laufzeit-Effizienz vs. Produktivität

Ruby on Rails

Ruby: Beispielcode

```
>> properties = ['object oriented', 'duck typed', 'productive', 'fun']
=> ["object oriented", "duck typed", "productive", "fun"]
>> properties.each {|property| puts "Ruby is #{property}."}
Ruby is object oriented.
Ruby is duck typed.
Ruby is productive.
Ruby is fun.
```

Ruby on Rails

Ruby: Installation

Installation via

- OS Paketmanager
- rvm
- rbenv

Ruby on Rails

Ruby: Programming Model

```
>> 4
=> 4
>> 4.class
=> Fixnum
>> 4 + 4
=> 8
>> 4.methods
=> ["inspect", "%", "<<", "singleton_method_added", "numerator", ...
"*", "+", "to_i", "methods", ...
]
```

Alles ist ein Objekt

Ruby on Rails

Ruby: Programming Model

```
>> x = 4
=> 4
>> x < 5
=> true
>> x <= 4
=> true
>> x > 4
=> false
>> false.class
=> FalseClass
>> true.class
=> TrueClass
```

Alles ist ein Objekt

Ruby on Rails

Ruby: Programming Model

```
>> 4 + 'four'
TypeError: String can't be coerced into Fixnum
from (irb):51:in `+'
from (irb):51
>>
=>
>>
=>
4.class
Fixnum
(4.0).class
Float
>> 4 + 4.0
=> 8.0
```

Duck Typing

Ruby on Rails

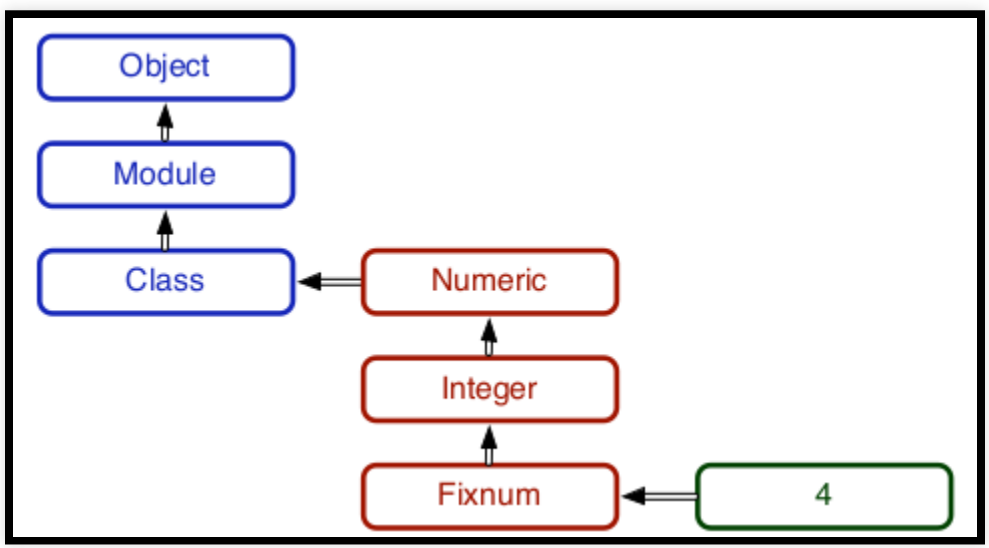
Ruby: Programming Model

```
If it walks like a duck and quacks like a duck, it's a duck.
```

- Dynamische Typisierung: Zur Ausführungszeit interpretiert
- Starke Typisierung: Typsicherheit

Ruby on Rails

Ruby: Metamodel



Ruby on Rails: Geschichte

- 2004: Entwickelt als Basis für *Basecamp*
- Version 1.0 (2005)
- Version 1.1 (2006) - Scripting Engines, Performance
- Version 1.2 (2007) - REST Support, MIME-type routing, UTF-8
- Version 2.0 (2007) - REST als Default
- Version 2.1 (2008) - Named Scopes, Migrationen mit Timestamp
- Version 2.2 (2008) - Internationalisierung, Threadsicherheit
- Version 2.3 (2009) - Template Engine
- Version 3.0 (2010) - Modularisierung: Einzelne Teile verwendbar
- Version 3.1 (2011) - Asset Pipeline
- Version 4.0 (2013)
- Version 5.0 (2016)
- Version 5.2 (2018)

Ruby on Rails: Bestandteile

- Active Support: Ruby-Erweiterungen von Rails
- Active Record: Objektabstraktionsschicht (ORM)
- Action Pack: Request-Behandlung und Response-Ausgabe.
- Action View: Templates
- Action Mailer: E-Mail-Versand und -Empfang
- Active Resource: Routing, REST, XML-RPC

Ruby on Rails: Grundlagen

- Basiert auf Ruby
- Model-View-Controller Architektur
- „Don’t repeat yourself“
- „Convention over Configuration“
- Scaffolding
- Datenbankmigrationen

„Don’t repeat yourself“

- Jede Information sollte nur ein einziges Mal vorhanden sein
- z.B. ActiveRecord liest das DB-Scheme direkt aus der DB
- z.B. Rails erstellt für das Model automatisch Getter- und Setter-Methoden
- Vorteil: Informationen werden nicht inkonsistent wenn eine Stelle verändert wird

Schichten: Model

- Typischerweise basierend auf einer relationalen Datenbank
- Zugriff mit Hilfe von ActiveRecord hergestellt (ORM-Schicht)
- --> Klasse <> Tabelle
- --> Attribut <> Spalte
- --> Objekt <> Zeile
- Support für: SQLite, DB2, Informix, MySQL, Oracle, PostgreSQL, Microsoft SQL Server, Sybase, ...
- Alternativen: Sequel, Datamapper, XML, ...

Ruby on Rails: Grundlagen

Schichten: Controller

- Basiert auf *ActionController*, Bestandteil von *ActionPack*
- Ein Controller kapselt die Geschäftslogik im Model von der Darstellung der View
- Ruby-Code

Schichten: View

- Präsentationsschicht *ActionView*, Bestandteil von *ActionPack*
- Diverse Ausgabeformate/Template-Engines werden unterstützt:
- -> HTML. ERB, Slim, Haml, Sass, CoffeeScript, ...
- -> XML – z. B. für XHTML und Web Services
- -> JSON
- -> JavaScript – RJS-Templates
- -> Binärdaten

Node.js

Node.js

Node.js ist eine serverseitige Plattform für Netzwerk-Anwendungen, die auf der Google Chrome JavaScript Engine (V8 Engine) basiert.

- easily build fast and scalable network applications
- event-driven, non-blocking I/O model
- lightweight, efficient, perfect for data-intensive real-time applications
- large JavaScript Library

Node.js

Node.js = Runtime Environment + JavaScript Library

https://github.com/nodejs/node

Node.js

Releases

- Current: Released from active development branches of this repository
- LTS: Releases that receive Long-term Support, with a focus on stability and security
- Nightly: Versions of code in this repository on the current Current branch

Node.js

Chrome V8

Google's high performance, open source, JavaScript engine.

- Open source, high-performance JavaScript engine
- Written in C++
- Used in Google Chrome
- It implements ECMAScript as specified in ECMA-262, 3rd edition
- Runs on Windows XP or later, Mac OS X 10.5+, and Linux systems that use IA-32, ARM or MIPS processors
- V8 can run standalone, or can be embedded into any C++ application

Node.js

Anwendungsfälle

- I/O bound Applications
- Data Streaming Applications
- Data Intensive Real time Applications (DIRT)
- JSON APIs based Applications
- Single Page Applications

Node.js

Grenzen

- CPU intensive applications

Node.js

REPL: Read Eval Print Loop

- Read - Reads user's input, parse the input into JavaScript data-structure and stores in memory.
- Eval - Takes and evaluates the data structure
- Print - Prints the result
- Loop - Loops the above command until user press ctrl-c twice.

Node.js

REPL: Read Eval Print Loop

- REPL feature of Node is very useful in experimenting with Node.js codes and to debug JavaScript codes.
- REPL can be started by simply running node on shell/console without any argument as follows.

```
$ node
>
```

Node.js: NPM

Node Package Manager

- https://www.npmjs.com/
- Find, share, and reuse packages of code from hundreds of thousands of developers
- 3 million developers and thousands of companies use npm

Node.js: NPM

Statistiken (Stand 13.05.2016)

- 283,432 total packages
- 57,724,266 downloads in the last day
- 1,040,776,238 downloads in the last week
- 4,169,158,104 downloads in the last month

Node.js: NPM

Geschäftsmodell

- Open Source: FREE
- Private Account: $7 per user / month
- Orgs: $16 per user / month
- Enterprise: $2000 per year
- npm Enterprise Pro

Node.js

Event Driven Programming

- The functions which listens to events act as Observers
- When an event gets fired, its listener functions start executing
- Node.js has multiple in-built events available through the **events module** and **EventEmitter class**

Ruby on Rails

Spring Boot (moovel Group GmbH)

NodeJS

AngularJS

Microservices (Github)

AMQP

Twitter Bootstrap

Blockchain (Chainstep)

AngularJS

AngularJS ist ein clientseitiges JavaScript-Webframework für Single-page-Webanwendungen nach dem Model-View-ViewModel-Muster

AngularJS

- Clientseitige Generierung von HTML-Seiten
- Funktionalität ohne DOM-Manipulation via jQuery
- Datenvalidierung von Eingabeformularen als Funktionalität

AngularJS

Komponenten

Die Strukturierung eines Angular-Webclients erfolgt auf Basis von

- Modulen
- View-Templates
- Controllern
- Scopes
- Filtern
- Providern (Factory, Service)

AngularJS

Datenbindung

- Datenbindung nach dem MVVM-Muster: Einfache Synchronisation zwischen View und Anwendungslogik
- Deklarative Beschreibungen von Datenbindungen innerhalb der View
- Event-Schleife fängt jede Eingabe ab und aktualisiert ggf. Teile der View
- Nicht editierbare Daten können mittels One-Time-Binding von weiteren Aktualisierungen ausgeschlossen werden.

AngularJS

Mocking-Modul

- Standardfunktionalitäten wie $http können für Tests ersetzt werden
- Umsetzung von isolierten Testfällen

AngularJS

Controller

- Clientseitiges Model
- Controller werden zu einem Modul zusammengefasst
- Module werden mittels eines Dependency-Injection-Containers in die Applikation eingebunden
- -> Dabei wird die View mit dem Model verbunden
- -> Bidirektionale Datenbindung

AngularJS

Direktiven

- benutzerdefinierte HTML-Elemente und -Attribute
- Vordefinierte Direktiven sind am ng-Namensraum erkennbar
- Um Elemente auszuwählen, verwendet AngularJS ein integriertes jQuery Lite (jqLite)
- Wird jQuery in das HTML-DOM eingebunden, wird dieses statt jQuery Lite verwendet.

AngularJS

Interpolation

- Mit doppelten geschwungenen Klammern können JavaScript-Ausdrücke ins HTML eingebettet werden
- Mit dem Pipe-Operator | können Filter hinzugefügt werden

AngularJS

Services

- enthalten die Geschäftslogik
- binden externe Ressourcen wie z.B. REST-Webservices ein
- werden als Singleton instanziiert
- können selbst implementiert werden oder von Dritten eingebunden werden (Beispiele $http und $resource)

AngularJS

Routen

- Zuordnung von URLs zu Views mittels *ngRoute*
- Unterschiedliche HTML Seiten können mittels *ng-view* nachgeladen werden.
- Eine *ng-view* Direktive pro Seite
- Das *$location* Objekt erlaubt die direkte Verarbeitung der Browser-URL

AngularJS

Vorteile

- Testbarkeit
- Refaktorisierbarkeit
- Wiederverwendbarkeit
- Verwendung das Javascript Datentypen

AngularJS

Nachteile

- Erstmal nur clientseitig
- Breaking Changes von V1 auf V2
- Performance
- Relativ komplex -> Viele Möglichkeiten Dinge falsch zu machen

AngularJS

Minimales Beispiel

Angular minimales Beispiel

AngularJS

Komplexeres Beispiel

- TodoList Beispiel
- Beispiel Code [github]

AngularJS

Quellen

- https://angularjs.org
- https://github.com/angular/angular.js
- https://docs.angularjs.org/guide

Fragen?

Ein Praxisbericht

Why a service oriented architecture is not the holy grail...

Vorbereitung auf Klausuraufgaben

- Was ist Dependency Injection?
- Nennen Sie drei Beispiele für Crosscutting Concerns
- Wo wird NodeJS typischerweise eingesetzt?
- Was ist das grundlegende Konzept einer NodeJS Anwendung?
- Wo sollte NodeJS eher nicht eingesetzt werden?
- Für welche Art von Anwendungen wird AngularJS verwendet?

Fragen?

Unterlagen: ai2018.nils-loewe.de