# Node.js Handout

## Bananengang

Sandra Kindler, Alexander Novikov, Amin Uzdenov, Joey Ruprecht

**Allgemeines**
Node.js ist JavaScript, welcher auf dem Server läuft. Es nutzt die V8 Engine von Google und kann eine viele Netzwerkverbindungen aufnehmen. Die Technologie ist weit verbreitet und basiert aktuell auf der Version 9.2. Node.js ist plattformübergreifend.

**Geschichte**
Node.js wurde 2009 von Ryan Dahl erfunden. Er wollte eigentlich eine Dateiupload realisieren und hat dafür verschiedene Skript- und Programmiersprachen ausprobiert. Die Lösung fand er in JavaScript. Nach und nach wurde aus seinem Projekt mehr und es entstand die serverseitige Technologie.

**Vorteile**
- Basiert auf JavaScript
- Gut dokumentiert
- Große Community
- Weitreichende Basisfunktionen
- nonblocking IO
- Native Unterstützung von JSON

**Nachteile**
- Das Stoppen der gesamten Laufzeit bei Auftritt eines Fehlers

**Was kann man mit node.js machen?**
Man kann beispielsweise in nur 4 Zeilen einen HTTP Server realisieren, welcher "Hallo Welt" printet. Auch kann man DNS Server, Static file Server, WebChat-Application oder auch Online Spiele damit realisieren.

**Aspekte für NodeJS:**
- Große Open Source Community
- Verarbeitung asynchronischer Events, basierend auf Event Loops mit Hilfe von libuv
- Ressourcenschonend beim datenintensiven Einsatz als Webserver oder für bidirektionale Websocket-Verbindungen, optimiert für Netzwerkanwendungen
- Gleiche Sprache für Front- und Backend

**Aspekte gegen NodeJS:**
- Intensive serverseitige CPU Operationen nicht empfehlenswert
- Durch Performance nicht für enterprise level application frameworks geeignet

**Module:**
- Dasselbe wie Bibotheken in Java

- Eine Menge an Funktionen die man seiner Anwendung hinzufügen kann
- Um Module zu benutzen: require()
- Um eigene Module zu erstellen: exports()

**Best Practices:**
- Package JSON
- Einen Style Guide benutzen
- Sicherstellen, dass die App automatisch neustartet(mit Prozess-manager, wie PM2)
- Logging libraries benutzen
- Immer asynchrone Funktionen verwenden
- Alle Module am Anfang requiren
- Callbacks validieren
- Use strict

**Design Patterns:**
Singletons:
- Das Pattern beschränkt die Anzahl von Instanzen einer "Klasse" auf 1
- Sehr einfach einzubauen da require() das für uns übernimmt, denn egal wie oft wir ein Modul importieren, es gibt immer nur eine Instanz davon
- Deshalb ist es das wahrscheinlich meistbenutzte Design-Pattern in Node.js

Observers:
- Ein Objekt hat eine Liste von Observern, welche benachrichtigt werden, sobald sich der Zustand des Objekts ändert
- Es gibt in Node.js ein natives Modul "events" und darin gibt es einen Eventemitter womit sich das Oberver-Pattern sehr einfach umsetzen lässt

Middleware:
- Der Output der einen Funktion ist der Input der nächsten, es wird mit einem next() weitergeschoben
- Sehr verbreitet bei Frameworks wie Express und Koa

**Event Loop:**
- Alle Anfragen für Input/Output Aufgaben und Callbacks mit einem Thread
- Nutzung von Worker Threads zur Benachrichtigung des Hauptthreads
- Keine Blockierung des Hauptthreads

**Einsatzgebiete:**
- JSON-basierte REST-Dienste
- Streaming
- Web-Echtzeit
- Single-Page-Anwendungen

**Alternativen zum Beispiel:**
- EventMachine for Ruby
- libuv for C
- Vert.x  for java