ISEngineering

Technische Universität Berlin

Anwendungssysteme - Übung 2

Dominik Ernst, Jonathan Heiß

Agenda

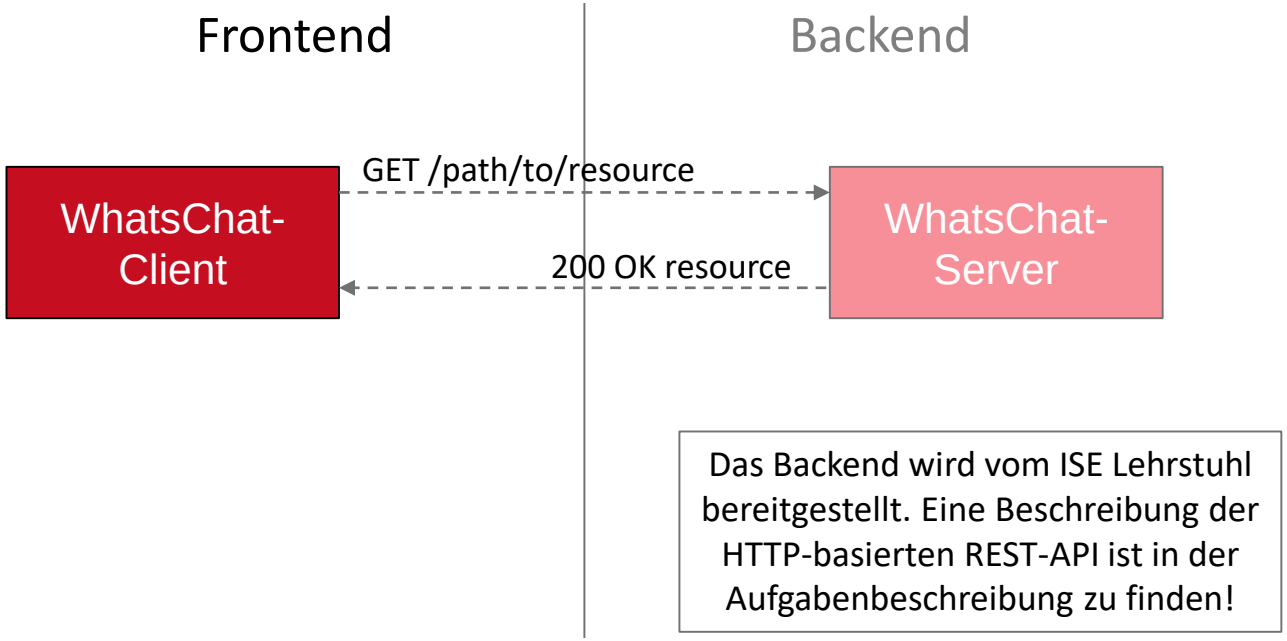
- Vorstellung “Musterlösung” A1
- Konzeptionelle Einführung
  - Maven
  - Spring Boot
- Demo: Beispielprojekt
- Aufgabenstellung A2

Recap: ChatAnwendung

Sie werden einen Browser-basierten **Client** für eine Chat-Anwendung implementieren!

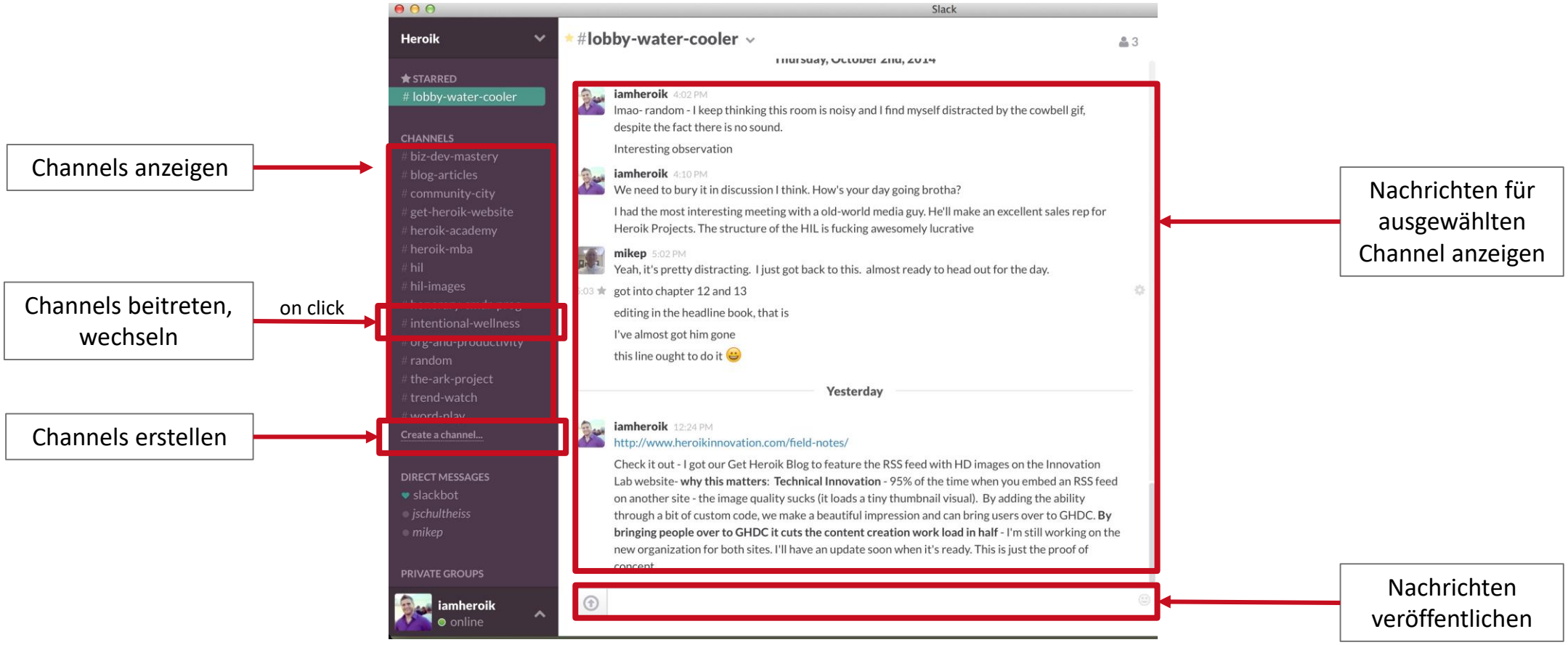
Technologien (Frontend):

- HTTP
  - Kommunikation mit dem Backend
- HTML/CSS
  - Strukturierung und Darstellung der Inhalte
- JavaScript
  - Nutzerinteraktion und Manipulation der HTML Seite

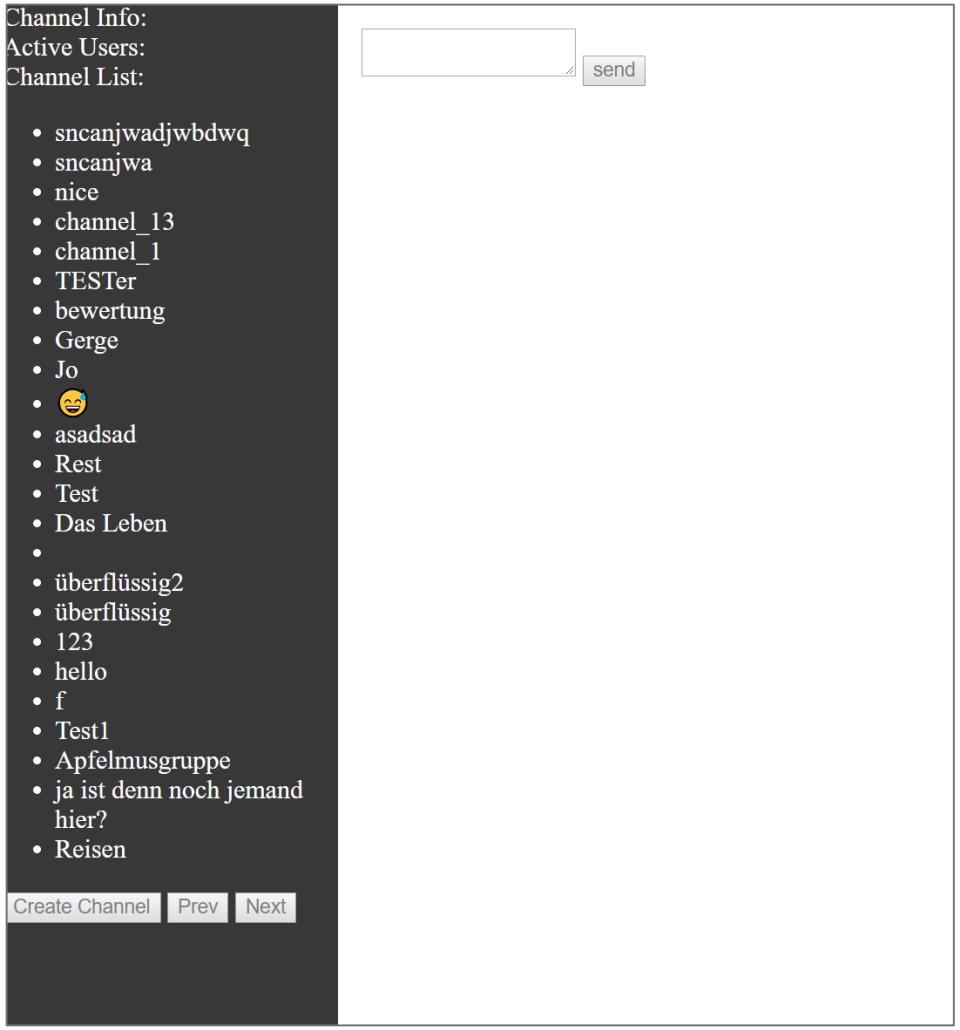


Das Backend wird vom ISE Lehrstuhl bereitgestellt. Eine Beschreibung der HTTP-basierten REST-API ist in der Aufgabenbeschreibung zu finden!

Recap: Funktionalität



User interface - HTML




```html
<!DOCTYPE html><html lang="en">
<head>... </head>
<body>
  <aside>
    <div id="channelInfo">Channel Info:</div>
    <div id="users">Active Users:</div>
    <div>Channel List:
      <ul id="channels"></ul>
      <button id="createChannel">Create Channel</button>
      <button id="prev">Prev</button>
      <button id="next">Next</button>
    </div>
  </aside>
  <main>
    <div id="sendMessages">
      <textarea id="messageText"></textarea>
      <button id="sendButton" >send</button>
    </div>
    <div id="messages"></div>
  </main>
</body>
</html>
```

Global Variables and Initialization

```
//constant variables
const url = 'http://34.243.3.31:8080';
const token = ‘abc123dEF456'

//global variables
var activeChannel;
var username = 'user_1’;
var lastSeenTimestamp= '1970-01-01T00:00:00Z’;
var channelPage = 0;
```

```
//initialize
$(document).ready(function() {
getChannels();
document.getElementById('createChannel').addEventListener('click', createChannel);
document.getElementById('sendButton').addEventListener('click', sendMessage);
document.getElementById('prev').addEventListener('click', () => {updatePage('prev')} );
document.getElementById('next').addEventListener('click', () => {updatePage('next')});
//(...)
});
```

1. Channels Erstellen

```
function createChannel(){
//get user input: channel name and topic
  let chName = prompt("Please enter the channel name:", "channel_1");
  let chTopic = prompt("Please enter the channel topic:", "topic_1");
//construct POST Request
  $.ajax({
    url: url + '/channels’,
    type: 'POST’,
    headers: {'X-Group-Token': token},
    contentType: 'application/json’,
    data: JSON.stringify( { "name": chName, "topic": chTopic } ),
    success: function() { alert('Channel added!'); }
  })
}
```

2. Channels Darstellen

```
function getChannels(){
//construct GET Request
 $.ajax({
  url: url + '/channels?page=‘+ channelPage +'&size=50’,
  type: 'GET’,
  dataType: 'json’,
  headers: {'X-Group-Token': token},
//render obtained JSON in HTML
  success: function (json){
    document.getElementById("channels").innerHTML="";
    $.each((json._embedded.channelList), function (i, item) {
      let newListElement = document.createElement('li’);
      newListElement.innerHTML = item.name;
      newListElement.addEventListener('click', () => {joinChannel(item.id)} );
      document.getElementById('channels').appendChild(newListElement);
 });}})}
```

```
setInterval( () => {getChannels()}, 10000);
```

```
function updatePage(reference){
  if(reference === 'next’){
  channelPage ++;
  }
  if(reference === 'prev’){
    if(channelPage > 0){
    channelPage --;
    }
  }
  getChannels();
}
```

3. Channels Beitreten

```
function joinChannel(id){
  activeChannel = id;
  username = prompt("Please enter your
user name:", “user_1");
  getChannelInfo(id);
  getUsers(id);
  getInitialMessages(id);
}
```

```
function getChannelInfo(id){
//construct GET Request
$.ajax({
  url: url + '/channels/' + id,
  type: 'GET’,
  headers: {'X-Group-Token': token},
  dataType: 'json’,
//render obtained JSON in HTML
  success: function(json){
    document.getElementById('channelInfo').innerHTML = '';
    let chName = document.createElement('div’);
    chName.innerHTML = 'Channel Name: ' + json.name;
    document.getElementById('channelInfo').appendChild(chName);
    let chTopic = document.createElement('div’);
    chTopic.innerHTML = 'Channel Topic: ' + json.topic;
    document.getElementById('channelInfo').appendChild(chTopic);
}})}
```

```
function getUsers(id){
if(id != undefined){
$.ajax({
    url: url + '/channels/' + id + '/users’,
    type: 'GET’,
    headers: {'X-Group-Token': token},
    dataType: 'json’,
    success: function(json){
      document.getElementById('users').innerHTML=‘’;
      $.each((json), function(i, item){
       let user = document.createElement('div’);
       user.innerHTML=item;
       document.getElementById('users').appendChild(
user);
})}})}}
```

```
setInterval(()=>{getUsers(activeChannel)}, 10000);
```

4. Nachrichten anzeigen (ohne timestamp)

```
function getInitialMessages(id){
//construct GET Request
$.ajax({
  url: url + '/channels/' + id + '/messages’,
  type: 'GET’,
  headers: {'X-Group-Token': token},
  dataType: 'json’,
//render obtained JSON in HTML
  success: function (json){
if( json._embedded != undefined){
    document.getElementById('messages').innerHTML = '';
    $.each((json._embedded.messageList), function (i, message) {
      let messageDiv = constructMessageDiv(message);
      document.getElementById('messages').append(messageDiv);
    })
lastSeenTimestamp = json._embedded.messageList[0].timestamp;
}}}})}
```

```
function constructMessageDiv(jsonMessage){
  let message = document.createElement('div’);

  let author = document.createElement('div’);
  author.innerHTML = jsonMessage.creator;
  message.appendChild(author);

  let content = document.createElement('div’);
  content.innerHTML = jsonMessage.content;
  message.appendChild(content);

  let timestamp = document.createElement('div’);
  timestamp.innerHTML =
calcTimestamp(jsonMessage.timestamp);
  message.appendChild(timestamp);

return message;
}
```

4. Nachrichten anzeigen (mit timestamp)

```
function updateMessages(id){
if(id != undefined){
//construct GET Request
$.ajax({
  url: encodeURI(url + '/channels/' + id + '/messages?lastSeenTimestamp=' + lastSeenTimestamp),
  type: 'GET’,
  headers: {'X-Group-Token': token},
  dataType: 'json’,
//render obtained JSON in HTML
  success: function (json){
    if( json._embedded != undefined){
      let newMessages = document.createElement('div');
      $.each((json._embedded.messageList), function (i, message) {
         let messageDiv = constructMessageDiv(message);
         newMessages.appendChild(messageDiv);
      }
      document.getElementById('messages’).prepend(newMessages);
  })
lastSeenTimestamp = json._embedded.messageList[0].timestamp;
}}}))}}
```

```
setInterval( () => {updateMessages(activeChannel)}, 1000);
```

5. Nachrichten verschicken

```
function sendMessage(){
//assign user input: message text
  let messageText = document.getElementById('messageText').value;
//construct POST Request
  $.ajax({
    url: encodeURI(url + '/channels/' + activeChannel + '/messages?lastSeenTimestamp=' + lastSeenTimestamp),
    type: 'POST’,
    dataType: 'json’,
    contentType: 'application/json’,
    data: JSON.stringify({"creator" : username, "content" : messageText}),
    headers: {'X-Group-Token': token},
//render JSON in HTML
    success: function (json){
      if( json._embedded != undefined){
      let newMessages = document.createElement('div’);
      $.each((json._embedded.messageList), function (i, message) {
        let messageDiv = constructMessageDiv(message);
        newMessages.appendChild(messageDiv);
      })
      document.getElementById('messages').prepend(messageDiv);
      lastSeenTimestamp = json._embedded.messageList[0].timestamp;
}}}})}
```

Anmerkungen

- Nachrichten, die den selben Zeitstempel haben werden in manchen Fällen doppel angezeigt
  - z.B. wenn mehrere Nachrichten pro Sekunde eingehen → Zeitstempel nur mit Sekundengenauigkeit

- Pagination beim Anzeigen der Nachrichten nicht gefordert
  - Es wurden statisch (maximal) die letzten 50 Nachrichten zurückgegeben

- Ergebnisse werden voraussichtlich bis Ende nächster Woche online gestellt!

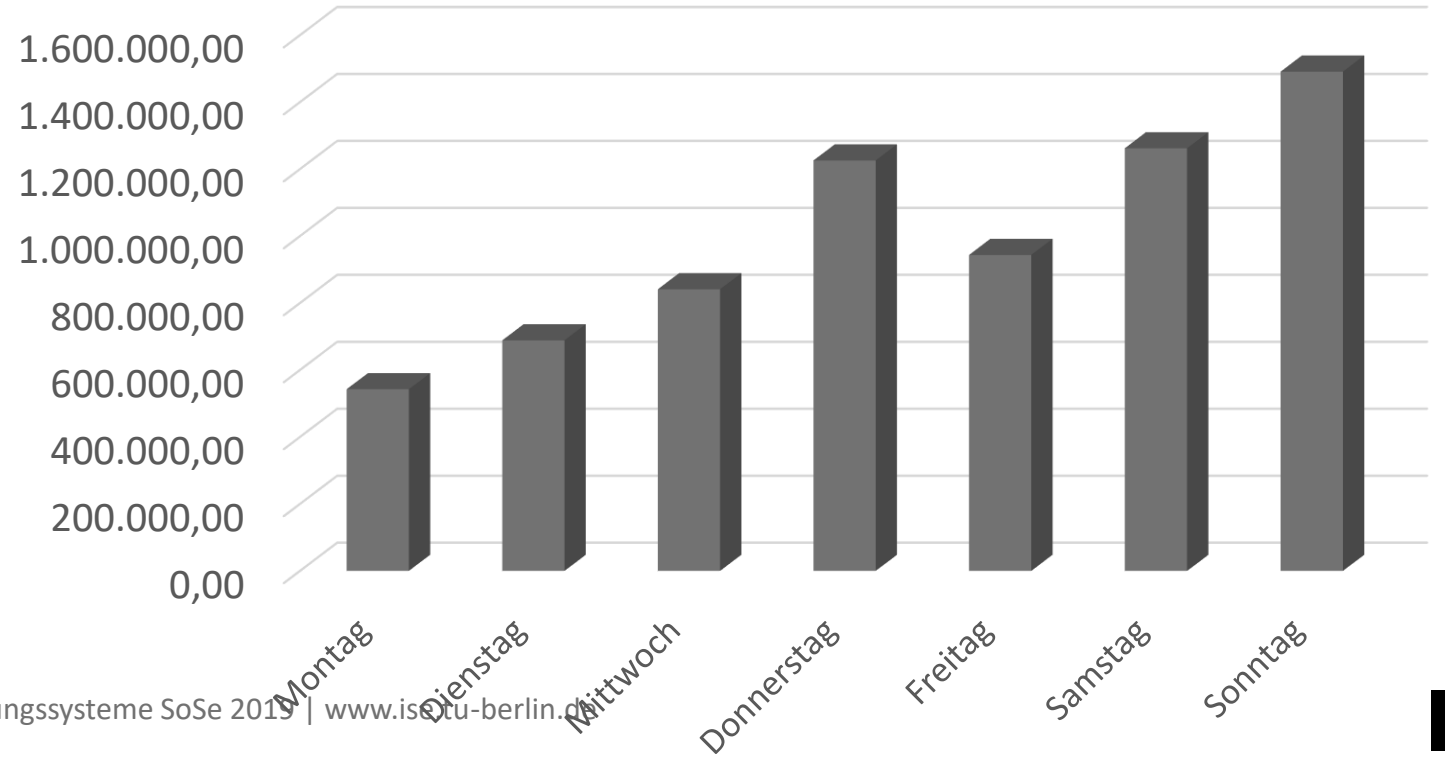
Statistics & Fun Facts

- Insgesamt 7,3 Mio Requests eingegangen (~800MB Logs)
  - Keine personenbezogenen Daten (IP-Addressen etc.), keine Inhalte von Nachrichten

Statistics & Fun Facts

- Serverseitig bestand ein Limit von 3req/s*Gruppe
- Gruppen-Awards (anonym!)
  - *Spammers* (höchste Anzahl POST requests): 2552
  - *Team DDOS* (höchste Anzahl requests insgesamt): 512.874

- “Arbeitswoche”



Übungsaufgabe 2

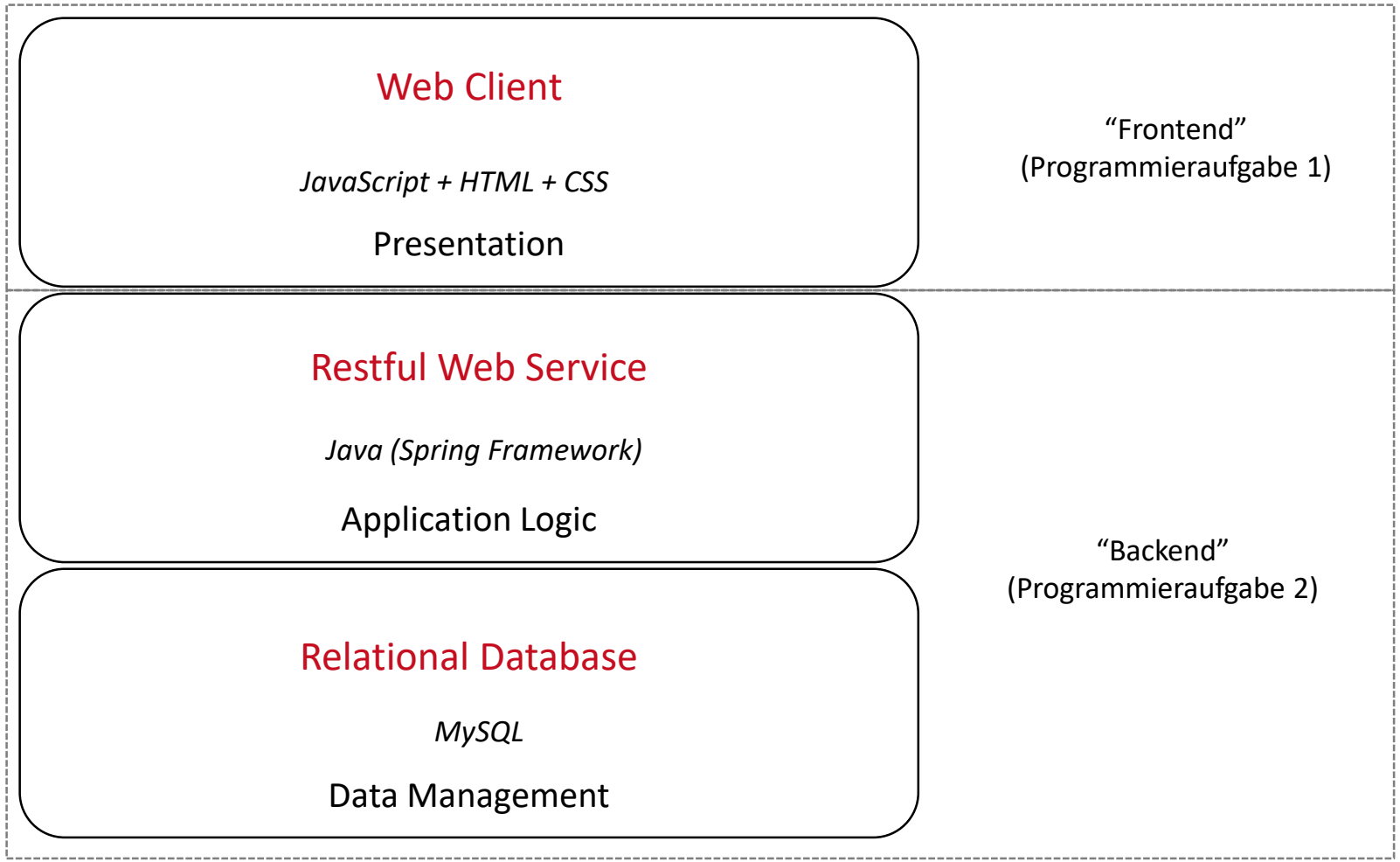
Lernziele

- In der Lage sein, mit Java einfache Web-Anwendungen zu entwickeln
  - Hands-on development
  - Technischen Hürden, bzw. teilweise offene Aufgabenstellung sind Absicht

- Praxisnahes Studium (von Lat. “studere” = sich (selbst) bemühen)
  - Gruppenarbeit ermöglicht „divide & conquer“
  - Google is your friend
  - “Nach welchen Begriffen würde man mit diesem Problem suchen?”

- Übergeordnet: “Full-Stack Application System Development”
  - Frontend- und Backendentwicklung
  - Eine Datenbank an die Anwendungslogik anbinden

Architektur



Restful Web Service

- REST = Architectural style of the Web
- URIs und HTTP als Technologiebausteine
  - Weitere Paradigmen (s. Vorlesungsfolien)

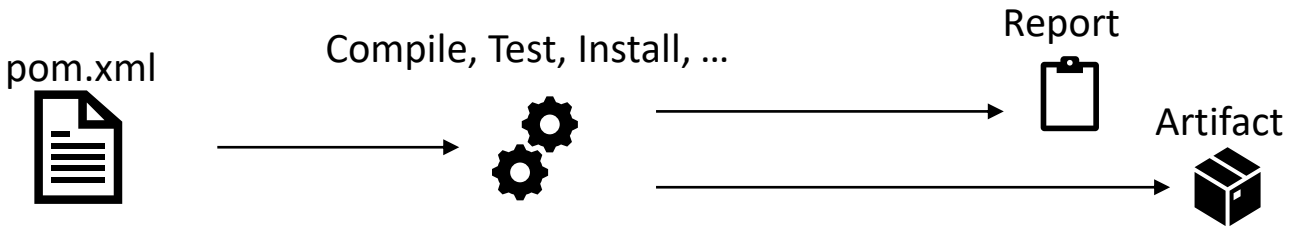
➔ Übungsaufgabe 2: RESTful Web-Service (Backend zu WhatsChat)

Ziel: einfach konfigurierbarer, losgelöst lauffähiger Service;
Implementierung in Java

➔ **Wir benutzen dafür Spring Boot. Eine *Maven Projekt* ist als Vorlage im ISIS verfügbar**

Recap: Maven

- Apache Maven = „Software-Project Management Tool“
  - Used as a build tool for Java artefacts
  - Centered around the “POM” (Project Object Model) in which dependencies and other characteristics of an artifact are described
  - Concept of lifecycle-phases: compile, test, package, install, deploy, …
  - Relies on repositories to find and download dependencies



- Define dependencies
- Parameters for build and test phases
- Versioning

- Download dependencies from repositories
- Run tests in specified order
- Create runnable artifacts, i.e. *.jar or *.war archives

- http://maven.apache.org/

Maven – POM File

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>tub.anwsys.chat</groupId>
  <artifactId>server</artifactId>
  <version>1.0</version>

  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.1.4.RELEASE</version>
  </parent>

  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
      <version>${spring-boot.version}</version>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-data-jpa</artifactId>
      <version>${spring-boot.version}</version>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-hateoas</artifactId>
      <version>${spring-boot.version}</version>
    </dependency>
    <dependency>
      <groupId>com.h2database</groupId>
      <artifactId>h2</artifactId>
      <version>${h2.version}</version>
    </dependency>
  </dependencies>

  <build>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
      </plugin>
    </plugins>
  </build>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <spring-boot.version>2.1.4.RELEASE</spring-boot.version>
  </properties>
</project>
```

Maven – POM File

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>

   <groupId>tub.anwsys.chat</groupId>
   <artifactId>server</artifactId>
   <version>1.0</version>

   <parent>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-parent</artifactId>
      <version>2.1.4.RELEASE</version>
   </parent>

   <dependencies>
      <dependency>
         <groupId>org.springframework.boot</groupId>
         <artifactId>spring-boot-starter-web</artifactId>
         <version>${spring-boot.version}</version>
      </dependency>
      <dependency>
         <groupId>org.springframework.boot</groupId>
         <artifactId>spring-boot-starter-data-jpa</artifactId>
         <version>${spring-boot.version}</version>
      </dependency>
```

POM-Files = XML

Identifikation des Artefakts zur Ablage im Repository

Hierarchische Einordnung (+Erbschaft von Dependencies)

Konkrete Dependency

Maven Lifecycle

- Vollständiges Diagramm: https://github.com/dtonhofer/diagrams/tree/master/Maven_Lifecycle
- Wichtige Phasen des “default”-lifecycles:

- Das Ausführen von Phasen ist zusätzlich mit sog. “Goals” verknüpft, die wiederum unterschiedliche Plugins ausführen können.

Spring Boot

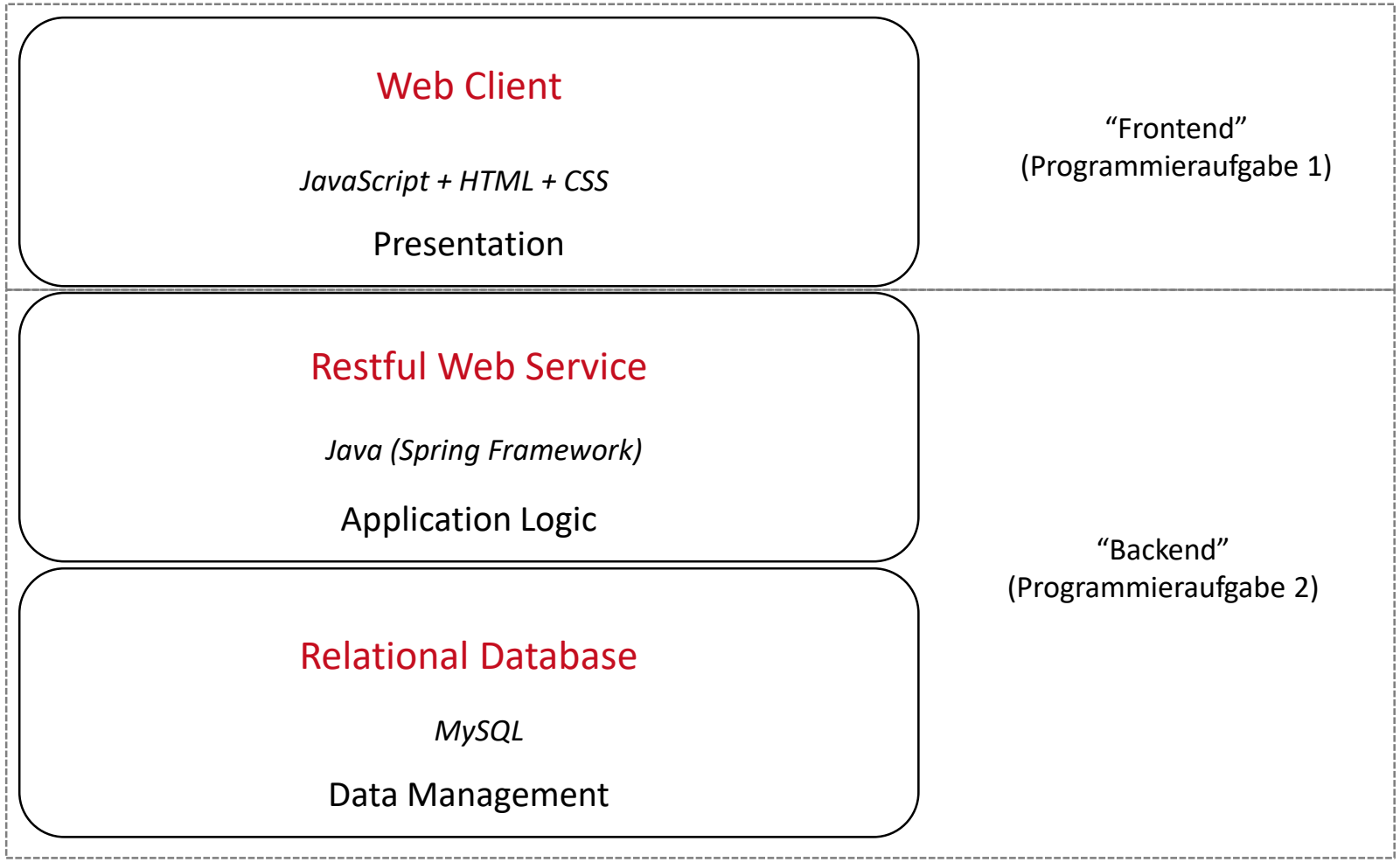
- Basis von Spring Boot ist das *Spring Framework*
  - Spring Framework ist ein Open-Source Framework für Java, das das Ziel hat die Konfiguration und das Ausführen von komplexen Anwendungen zu vereinfachen
  - Kernkonzepte: Inversion of Control (IOC) via (Runtime) Resource Injection + Aspect Oriented Programming (AOP)

- Spring Boot vereinfacht das Erstellen von Standalone Java Web-Programmen
  - Embedded Tomcat Web Server ➔ .jar-Datei enthält eine Implementierung des Web Servers
  - „Convention over Configuration“: brauchbare Standardwerte für Konfiguration, Änderung bei Bedarf

- Wir verwenden mehrere Spring Boot Komponenten
  - spring-boot-starter-web: Embedded Webserver + Web Framework für REST-APIs
  - spring-boot-starter-data-jpa: Adaptierung der Java Persistence API, die Hibernate als JPA Implementierung nutzt

- Spring ist kein Standard (wie z.B. JPA), sondern eine Sammlung von Frameworks

Architektur



Datenbank - Setup

- Jedes nicht-triviale Anwendungssystem speichert Daten
- Für das Backend von WhatsChat nutzen wir eine Relationale Datenbank: MySQL

Zwei Varianten zur lokalen Nutzung auf ihrem Rechner:

1. Lokale Installation: https://dev.mysql.com/downloads/mysql/
   - "Server only" bei der Installation auswählen genügt
   - Während der Installation werden Port und Root Password festgelegt, diese entsprechend wählen!
   - Über die MYSQL-Shell muss noch eine Datenbank angelegt werden

2. Als Docker Container (benötigt Installation von Docker [1]):
   - "mysql" ist als Container Image im public repository verfügbar
   - One-liner: docker run -d -p 3306:3306 -e MYSQL_ROOT_PASSWORD=ThePass -e MYSQL_DATABASE=chatdb mysql

[1] https://hub.docker.com/search/?type=edition&offering=community

Demo

Programmieraufgabe 2

Aufgabenbeschreibung - Zusammenfassung

- PDF mit ausführlicher Aufgabenstellung im ISIS
- Projektvorlage im ISIS

- Aufgabe: WhatsChat Backend
  - Channels erstellen/abrufen
  - Nachrichten schicken / abrufen
  - User anzeigen

- **Deadline: 07.07.2019 (2,5 Wochen)**

- *Nächste Woche keine regulären Tutorien, stattdessen Fragestunde für technische und Verständnisprobleme.*

Links & Infos

- Spring REST Tutorial (short): https://spring.io/guides/gs/rest-service/
- Spring Data JPA Tutorial: https://spring.io/guides/gs/accessing-data-jpa/
- Spring REST HATEOAS (long): https://spring.io/guides/tutorials/rest/

- MySQL First Steps: https://dev.mysql.com/doc/mysql-getting-started/en/#mysql-getting-started-connecting
- Docker MySQL one-liner: docker run -d -p 3306:3306 -e MYSQL_ROOT_PASSWORD=ThePass -e MYSQL_DATABASE=chatdb mysql