PROGRAMMIERPROJEKT 2018

WEB-ANWENDUNGEN

Dr. Annette Bieniusa

ÜBERBLICK

- Was ist eine Web-Anwendung?
- Model-View-Controller Pattern
- Umsetzung für unser Projekt
  - Spark
  - SQL
  - SQL in Java

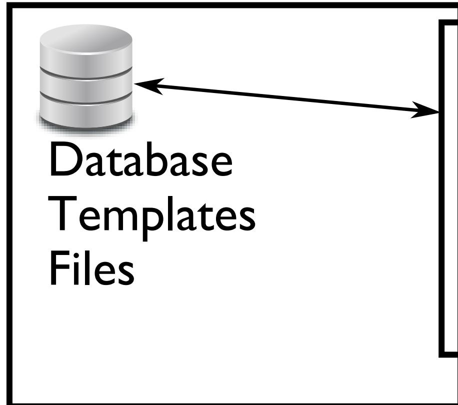
WEB-ANWENDUNG

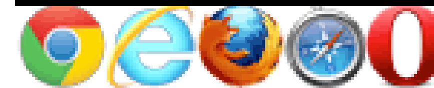
- **HTML** ist ein Format um **Inhalte** einer Seite zu beschreiben
- Inhalte grundsätzlich statisch
- **CSS** erlaubt **Gestaltung** der Inhalte, hauptsächlich statisch
- **JavaScript** erlaubt **dynamische** Änderungen am HTML
  - Aber: Kein direkter Austausch von Informationen zwischen Benutzern möglich

- Web-Anwendungen laufen auf **Servern** und kommunizieren mit dem Browser des Benutzers
- Austausch von Informationen zwischen Benutzern möglich
- Server **speichert Informationen** zur weiteren Verwendung
- **Client-Server** Prinzip

KLASSISCHE SICHT

- Server erzeugt HTML und sendet es an den Browser
- Browser zeigt HTML dem Benutzer an
- Benutzer gibt Informationen in den Browser ein
- Browser sendet Informationen an den Server
- Empfangene Informationen können in Generierung der HTML-Seite eingehen


Server
Database
Templates
Files
GET /index
200 <html>...</html>
POST /login
username=...
password=...
Client (Browser)
Login please
User
Password
Login

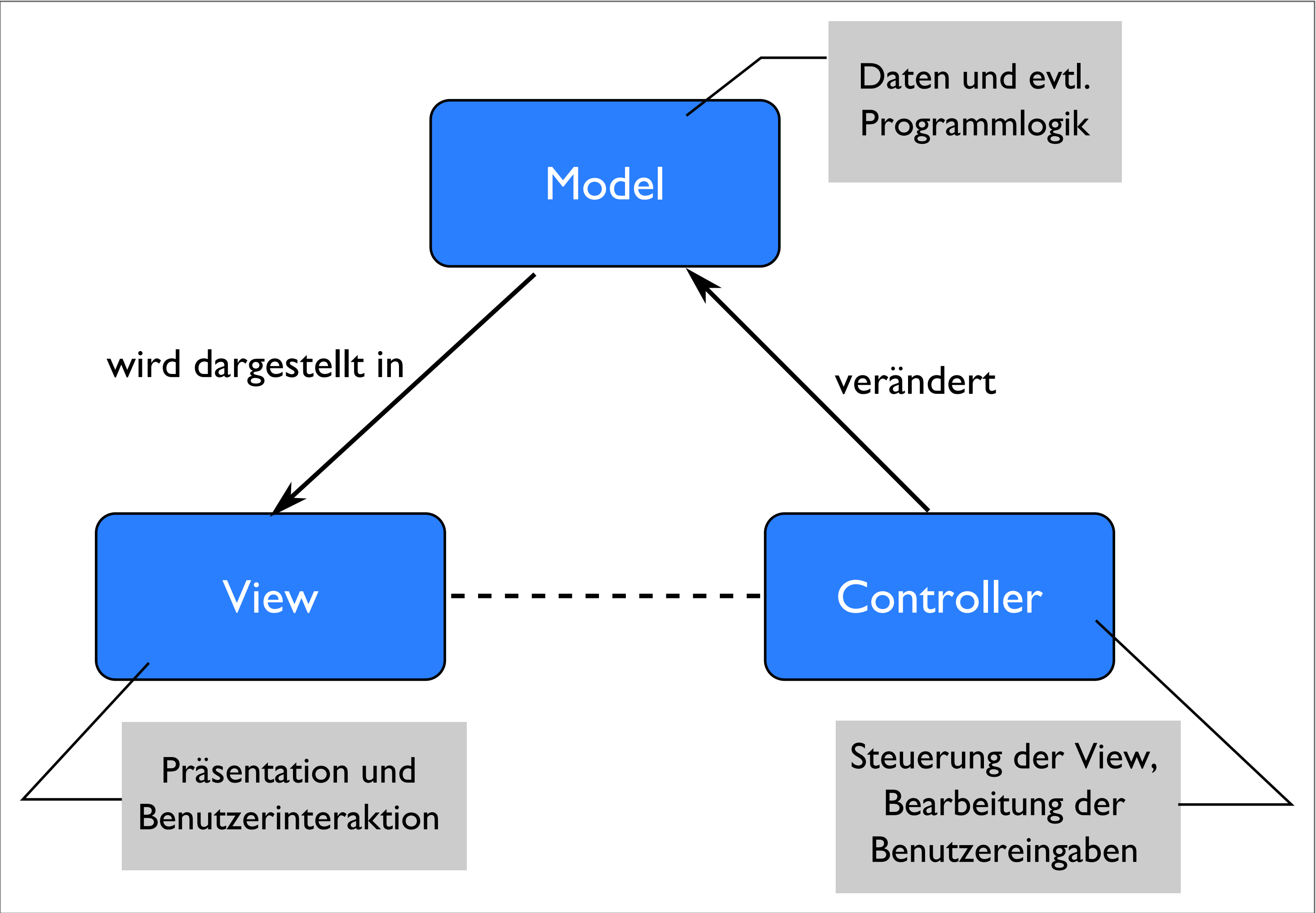


MODERNE SICHT

- Server stellt Daten als Service zur Verfügung (z.B. REST-Services)
- Browser verarbeitet Daten mit Hilfe von JavaScript Anwendungen (z.B. AngularJS)
- *Wir verwenden hier die klassische Sicht!*

MODEL-VIEW-CONTROLLER

- Muster zur **Strukturierung** von Software
  - Architekturmuster
  - Entwurfsmuster
- **Wiederverwendbarkeit** von Komponenten
  - Bsp.: Austausch der GUI (hier: des HTML-Codes)


Model
Daten und evtl. Programmlogik
wird dargestellt in
verändert
View
Controller
Präsentation und Benutzerinteraktion
Steuerung der View, Bearbeitung der Benutzereingaben

ZUORDNUNG VON FUNKTIONALITÄT

- Geschäfts-/Programm**logik** sollte Teil des Modells sein
  - **Testbarkeit!**
- **Validierung** von Benutzereingaben
  - Einfache Überprüfungen des Formats: View/Controller
  - Komplexere Überprüfungen: Modell
- Daten-**Formatierung** (z.B. Datumsformat)
  - Für unser Projekt: meist Controller

SPARK

WAS IST SPARK?

- Framework zur Erstellung von Web-Anwendungen in Java
- Einfach zu verwenden
- Integrierter Webserver

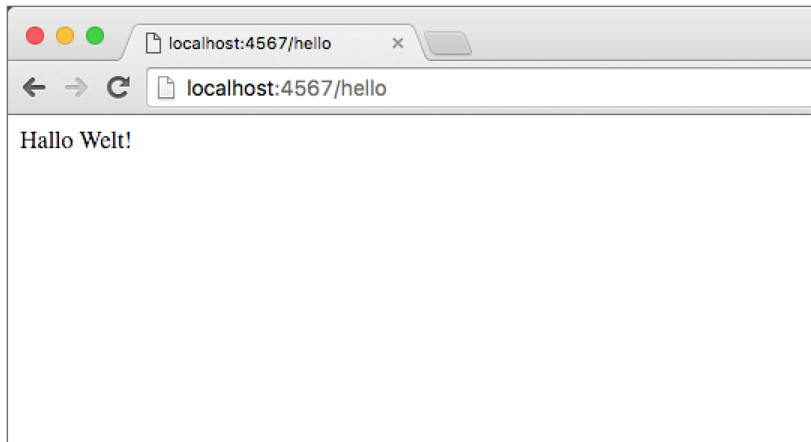
WIE FUNKTIONIERT SPARK?

1. Web-Anwendung bekommt Anfrage (engl. request) auf eine URL
2. Aufruf einer Java-Methode, welche HTML als `String` erzeugt
3. HTML wird (mit zusätzlichen Meta-Daten) als Antwort (engl. response) zurückgeschickt

```java
import static spark.Spark.*;
        import spark.*;

        public class Main {
          public static void main(String[] args) {
            get("/hello", new Route() {
              @Override
              public Object handle(Request request, Response response) throws Exception {
                return "Hallo Welt!";
              }
            });
          }
        }
```

Anfragen an `/hello` werden durch die `handle` Methode behandelt und geben `Hallo Welt!` zurück.


localhost:4567/hello
localhost:4567/hello
Hallo Welt!

In Java 8 kann der Code zu

```
import static spark.Spark.*;
        import spark.*;

        public class Main {
          public static void main(String[] args) {
            get("/hello", (request, response) -> "Hallo Welt!");
          }
        }
```

verkürzt werden.

Auch Einfügen von dynamischen Inhalten möglich:

```
get("/hello", (request, response) -> {
        Date now = new Date();
        DateFormat dateFormat = SimpleDateFormat.getDateInstance();
        DateFormat timeFormat = SimpleDateFormat.getTimeInstance();
        return "<h1>Hallo Welt!</h1>" +
           "<p>Datum: " + dateFormat.format(now) +
           " Uhrzeit: " + timeFormat.format(now) + "</p>";
         });
```

localhost:4567/hello
localhost:4567/hello
Hallo Welt!
Datum: 08.04.2016 Uhrzeit: 15:25:45

Mehr Informationen zu Spark, insbesondere zu den Themen

- Welche Methoden gibt es auf `Request`?
- Welche Methoden gibt es auf `Response`?
- Wie erzeuge und lese ich Cookies?
- Wie verwalte ich eine Session?
- Wie leite ich auf eine andere Seite um?

unter **http://sparkjava.com/documentation.html**

HTML TEMPLATES

- HTML-`String`s im Java Code werden schnell unübersichtlich
- Trennung zwischen HTML und Code erwünscht wegen
  - Editorunterstützung
  - Einfacher Vorschau
  - Möglichkeiten der Kooperation

TEMPLATE ENGINES

- Template (z.B. HTML-Datei) mit Platzhaltern
- Engine nimmt Template und "Modell" und füllt Platzhalter
- Template in separater Datei und Modell kann in Java erzeugt werden

*Wir verwenden hier **Mustache**.*

In Datei `hello.mustache`:

```
<html>
  <head>
    <title>Hallo</title>
  </head>
  <body>
    <h1>Hallo Welt!</h1>
    <p>Datum: {{datum}}  Zeit: {{zeit}}</p>
  </body>
</html>
```

In der `main`-Methode:

```
get("/hello", (request, response) -> {
  Map<String, Object> modell = new HashMap<>();
  Date now = new Date();

  DateFormat dateFormat = SimpleDateFormat.getDateInstance();
  DateFormat timeFormat = SimpleDateFormat.getTimeInstance();

  modell.put("datum", dateFormat.format(now));
  modell.put("zeit", timeFormat.format(now));

  return new ModelAndView(modell, "hello.mustache");
  }, new MustacheTemplateEngine());
```

Man kann auch auf Attribute eines Objekts zugreifen:

```
public class Person {
  private String name;
  private int age;
  public Person(String name, int age) {
    this.name = name;
    this.age = age;
  }
  public String getName() { return this.name; }
  public int getAge() { return this.age; }
  ...
}
```

```
// in get
modell.put("myperson", new Person("Hans", 45));
```

Im Template:

```
... {{myperson.name}} ist {{myperson.age}} ...
```

Über Listen kann iteriert werden:

```
// in get
List<String> personen = new ArrayList<>();
personen.add("Julia");
personen.add("Maria");
personen.add("Herbert");
modell.put("personen", personen);
```

Template:

```
{{#personen}}
- {{.}}
{{/personen}}
```

Ausgabe:

```
- Julia
- Maria
- Herbert
```

Bedingte Formatierung:

```
Map<String, Object> model = new HashMap<>();
model.put("deutsch?", true);
...
```

Template:

```
{{#deutsch?}}
<h1>Hallo Welt!</h1>
{{/deutsch?}}
{{^deutsch?}}
<h1>Hello World!</h1>
{{/deutsch?}}
```

Ausgabe:

```
<h1>Hallo Welt!</h1>
```

Weitere ausführliche Informationen zu Mustache finden Sie auf der Homepage unter

http://mustache.github.io/

und in der Spezifikation

https://github.com/mustache/spec

SQL

RELATIONALE DATENBANKEN

- Ermöglicht Speichern von Daten einer Anwendung
- Einteilung der Daten in **Relationen** (Tupel von Attributen)
- Abspeichern der Relationen in **Tabellen**
- Nur **atomare Attribute** in der Tabelle
- **Primärschlüssel** zur eindeutigen Identifikation einer Zeile

MatrikelNr	Vorname	Nachname
367891	Paul	Muster
378912	Horst	Müller
389123	Lisa	Schmidt

- Attribute: MatrikelNr, Vorname, Nachname
- Primärschlüssel: MatrikelNr
- Typen
  - MatrikelNr: Zahlen
  - Vorname und Nachname: Zeichenketten

TABELLE ANLEGEN

```
CREATE TABLE studenten
(MatrikelNr    INT          PRIMARY KEY,
Vorname       CHAR(100),
Nachname      CHAR(100))
```

- `CREATE TABLE` erzeugt eine Tabelle
- `PRIMARY KEY` markiert den Primärschlüssel
- `INT` steht für ganze Zahlen, `CHAR(100)` für Zeichenketten mit maximaler Länge 100

EINFÜGEN VON WERTEN

```
INSERT INTO studenten (MatrikelNr, Vorname, Nachname)
VALUES ( 367891, 'Paul', 'Muster' )
```

Man kann die Liste der Attribute weglassen, wenn man die Werte in der Reihenfolge der Attribute angibt.

LESEN VON WERTEN

```
SELECT Vorname, Nachname
FROM studenten
WHERE MatrikelNr = 367891
```

Mit `SELECT *` kann auch das gesamte Tupel ausgelesen werden.

Nicht auf Primärschlüssel beschränkt (z.B. `WHERE Vorname='Heinz' AND Nachname='Müller'`). Ergebnis kann auch mehrere Tupel umfassen.

LÖSCHEN VON TUPELN

```
DELETE FROM studenten
WHERE MatrikelNr = 367891
```

ÄNDERN VON TUPELN

```
UPDATE studenten
SET Vorname = 'Heinz'
WHERE MatrikelNr = 367891
```

VERBINDEN VON TABELLEN

MatrikelNr	Vorlesung	Note
389123	SE1	2,3
389123	Logik	4,0

- Ergebnisse von **Studenten**
- Nur gültig falls Student vorhanden

```
CREATE TABLE ergebnisse
(
MatrikelNr   INT,
Vorlesung    CHAR(100),
Note         CHAR(10),
FOREIGN KEY (MatrikelNr) REFERENCES studenten
)
```

ZUSAMMENFÜGEN BEIM AUSLESEN

```
SELECT S.MatrikelNr, S.Vorname, S.Nachname, ER.Vorlesung, ER.Note
FROM studenten S, ergebnisse ER
WHERE S.Vorname='Heinz' AND ER.MatrikelNr = S.MatrikelNr
```

- Ergebnisse von Studenten mit Vornamen Heinz
- Zusammenfügen (engl. join) über Attribute `MatrikelNr`
- Attribute von beiden Tabellen zugreifbar
- Alias für Tabellen, um weniger schreiben zu müssen

SQL IN JAVA

INTERAKTION MIT DER DATENBANK MITTELS JDBC

- Kommunikation mit Datenbank über **Verbindung** (`Connection`)
- Absetzen von Befehlen über **Anweisungen** (`Statement`)
- Anweisungen mit Lücken und automatischem Maskieren/Escapen (`PreparedStatement`)
- Iterieren über die Liste der Ergebnisse (`ResultSet`)

UPDATE UND QUERY

```
Connection con = ...;
        Statement stmt = con.createStatement();
        int n = stmt.executeUpdate("DELETE FROM studenten WHERE MatrikelNr=367891");
        // n enthaelt die Anzahl der geloeschten Zeilen
```

Für `INSERT`, `UPDATE`, `DELETE`, `CREATE TABLE`.

```
Connection con = ...;
        Statement stmt = con.createStatement();
        ResultSet res = stmt.executeQuery("SELECT * FROM studenten WHERE MatrikelNr=367891");
        while(res.next()) {
          int matriklnr = res.getInt("MatrikelNr");
          String vorname = res.getString("Vorname");
          String nachname = res.getString("Nachname");
          // ...
        }
```

Für `SELECT` mit Ergebnis.

DYNAMISCHE ANFRAGEN

```
public class Dynamic {
          public static void main(String[] args) {
            Connection con = ...;
            Statement stmt = con.createStatement();
            stmt.executeUpdate("DELETE FROM studenten WHERE MatrikelNr=" + args[0]);
          }
        }
```

Schlecht: java Dynamic "367891; DELETE FROM ergebnisse"

```
DELETE FROM studenten WHERE MatrikelNr=367891;
        DELETE FROM ergebnisse
```

Nebeneffekt: Alle Ergebnisse gelöscht!

PREPAREDSTATEMENT

```java
public class Dynamic {
  public static void main(String[] args) {
    Connection con = ...;
    PreparedStatement pstmt = con.prepareStatement(
      "DELETE FROM studenten WHERE MatrikelNr=?");
    pstmt.setInt(1, Integer.parseInt(args[0]));
    pstmt.executeUpdate();
  }
}
```

Vermeidet missbräuchliche Verwendung durch Escapen.

Auch `pstmt.setString(1, args[0])` würde zum Fehler führen, weil nicht typkorrekt.