

Web 2.0 Technologien 2

Kapitel 3:

Webserver-Frameworks: Django

Limitierungen der Low-Level-Entwicklung (1)

*Die Low-Level-Entwicklung mit PHP und SQL ist bzgl. **einer effizienten Softwareentwicklung** limitiert ...*

- **Problem 1: Funktionalität wird durch primitive Operationen realisiert**
 - z.B. Formularverarbeitung: Jedes DB-Attribut ...
 - ... muss bei der Formularausgabe erzeugt und ggf. vorbelegt werden
 - ... muss bei der Formulardatenverarbeitung geprüft und einem Datensatz-Attribut zugewiesen werden
 - Eine Erweiterung des Schemas führt zu vielen Anpassungen
- **Problem 2: Mischung von Darstellungs- und Kontroll-Ebene führt zu unübersichtlichen Strukturen**
 - Es ist oft schwer zu erkennen, was die wesentlichen Kontrollabläufe sind und ob alle Fälle vollständig behandelt sind

Limitierungen der Low-Level-Entwicklung (2)

*Die Low-Level-Entwicklung mit PHP und SQL ist bzgl. **der Sicherung der Datenintegrität** limitiert ...*

- **Problem 3: Datenintegrität wird an vielen Punkten verteilt bzw. repliziert gesichert**
 - z.B. Feldlänge / Typ / Wertebereich eines Attributs eines editierbaren DB-Datensatzes ...
 - ... ist im DB-Schema (SQL) festgelegt
 - ... muss an HTML-Formular übergeben werden
 - ... muss bei POST-Daten geprüft werden
 - Eingabefehler sollten per Postback im Formular angezeigt werden
 - Änderung des Schemas führt dadurch zu vielen Anpassungen

Limitierungen der Low-Level-Entwicklung (3)

*Die Low-Level-Entwicklung mit PHP und SQL ist bzgl. **Schutz vor Sicherheitslücken** limitiert ...*

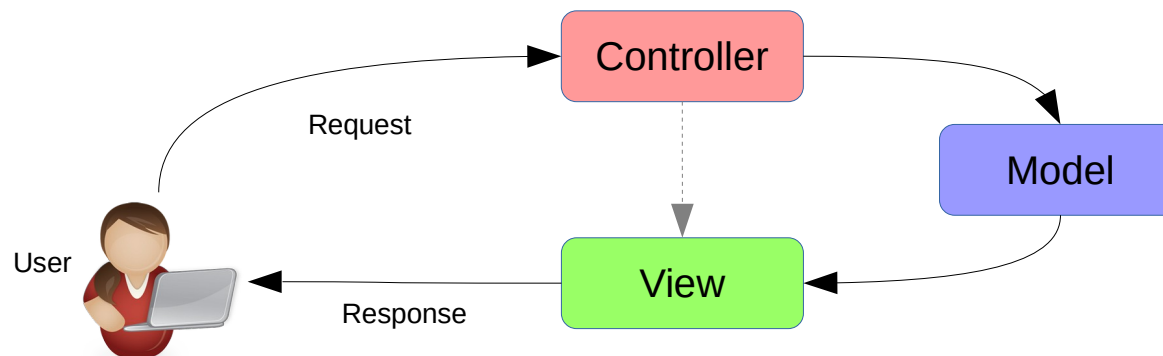
- **Problem 4:** Sicherheitsprobleme müssen oft manuell und überall im Code verteilt gelöst werden
 - vollständige Abschottung aller potentiellen Angriffsvektoren erforderlich
 - dies erfolgt auf sehr geringem Abstraktionsniveau, z.B.
 - SQL-Injections vermeiden bei textuellen Aufbau von SQL-Queries
 - XSS-Attacken vermeiden bei textuellem Aufbau von HTML-Ausgaben
- **Problem 5:** Mechanismen sind oft „**Unsafe by default**“
 - Sicherheit bekommt man nur, wenn man ***alles richtig*** macht

Anforderungen

- **Wir brauchen ...**
 - **Trennung** von Modell, Präsentation und Steuerung (→ **MVC**)
 - Klarere Struktur der Applikation
 - Keine Durchmischung von Applikationslogik und Darstellung
 - **Abstrakte** Schema-Definition (Modell)
 - Schema-Eigenschaften werden nicht wiederholt sondern zentral definiert und gesichert (und überall benutzt)
 - Implementierung von Routine-Abläufen auf **hoher Ebene**
 - Automatisch Änderungsformulare zum Schema erzeugen, Antworten prüfen und speichern
 - **Sicherer** Umgang mit unsicheren Daten (u.a. Präsentation)
 - **Escape-by-Default** für alle unsicheren Inhalte (**Safe-by-Default**)
 - Mechanismen zur **zuverlässigen** Fehler- und Ausnahmebehandlung
 - z.B. keine unkontrollierten Ausgaben an den Nutzer (mitten in der Webseite) im Fehlerfall
 - ...

Hintergrund: MVC

- Entwurfsmuster „**Model – View – Controller**“ (**MVC**)
 - Idee: Klare **Trennung** von **Modell**, **Präsentation** und **Steuerung**
 - Das **Modell** („**model**“) hält und pflegt die **Daten**
 - Unabhängig von View und Controller
 - Die **Präsentation** („**view**“) **stellt** die Daten dem Benutzer **dar**
 - Daten werden aus dem Modell gelesen und dargestellt
 - Die **Steuerung** („**controller**“) verwaltet Modell und Präsentation
 - Sie verarbeitet und kontrolliert Änderungen und steuert auch z.B. die Einhaltung von Benutzerrechten (wer darf was sehen oder ändern)



Hintergrund: MVC

- Entwurfsmuster „**Model – View – Controller**“ (**MVC**)
 - Es gibt **viele Varianten** und offene Punkte dieses Modells
 - Siehe https://de.wikipedia.org/wiki/Model_View_Controller
 - *Zur Vertiefung:*
 - Wie sind die Abläufe in einem LAMP-Server hier zuzuordnen?
 - Was davon haben wir bisher davon explizit architekturell abgegrenzt?
 - Siehe Abschnitt „[Serverseitige Webanwendungen](#)“
 - Wie sind die Abläufe des Systems Webserver + Webclient hier einzuordnen?
 - Siehe Abschnitt „[Zusammenspiel von Server und Browser bei Webanwendungen](#)“
 - Viele Web-Frameworks setzen das Konzept um
 - Allerdings oft mit sehr unterschiedlichen Sichtweisen
 - Django (s.u.) hat z.B. eine eher unorthodoxe Sicht auf den Begriff „**View**“:
 - <https://docs.djangoproject.com/en/4.2/faq/general/>
 - <https://django-book.readthedocs.io/en/latest/chapter01.html#the-mvc-design-pattern>

Lösungsansätze (1)

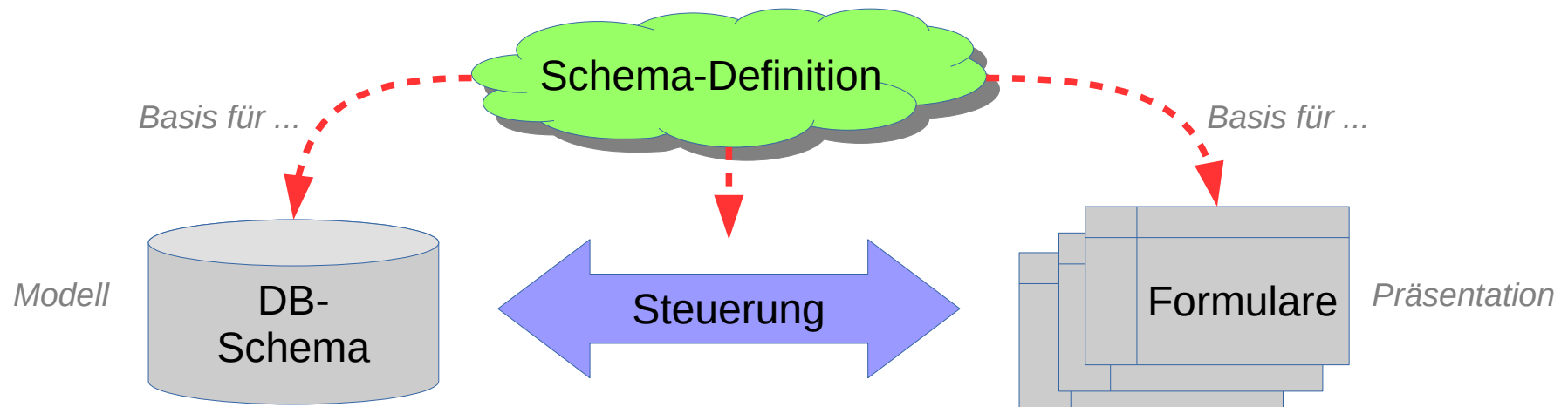
- **Trennung von Modell, Präsentation und Steuerung**

- Separate Kontrolllogik / Steuerung 
 - Klare und verständliche Strukturen bei der Anfrageverarbeitung
- Mechanismen zur sicheren Fehlerbehandlung
 - z.B. keine unkontrollierten Ausgaben an den Nutzer im Fehlerfall
- Nutzung von **Template-Engines** zur Präsentation 
 - Idee: Daten + **Template** → HTML-Response
 - Darstellungsaspekte (u.a. Design) werden im Template behandelt
 - Template- und Applikationsdesigner können getrennt / parallel arbeiten
 - Variable Designs durch umschaltbare Template-Sätze
- **Escape-by-Default** für alle unsicheren Inhalte
 - Tagging (interne Markierung / Klassifizierung) von Strings
 - unsicher / sicher / bereits escaped
 - vermeidet Injection-Probleme (XSS, HTML-, SQL-Injection)

Lösungsansätze (2)

- **Nur eine Schema-Definitionen für Datenablage (Modell) und Formularbehandlung**

- Beschreibung des Schemas auf hoher Abstraktionsebene



- Generierung von DB-Schema und Formularen aus dem selben **normativen Schema** (Das DB-Schema könnte auch als normatives Schema dienen.)
- Automatische Prüfung von Formulardaten auf Schema-Konformität
- Bietet konsistenten / effizienten Mechanismus zu Verarbeitungskette
DB → **Formular** → **Bearbeitung** → **Formular** → **DB**

Lösungsansätze (3)

- **Applikationsstruktur, die Routineabläufe automatisiert**
 - „**Konvention vor Konfiguration**“
 - Softwaredesign-Paradigma („*Don't Repeat Yourself*“)
 - http://de.wikipedia.org/wiki/Konvention_vor_Konfiguration
 - Hohes Abstraktionsniveau wo immer möglich, z.B.
 - Abstraktes Schema (einmal für Alles)
 - Abstrakte Filterung von Daten (keine komplizierten SQL-Joins)
 - Robuste URL-Analyse und -Synthese
 - Hierarchische HTML-Templates
 - Robustheit und Pflegbarkeit
 - Logging / Benachrichtigung des Administrators bei Fehlern
 - automatische Schema- und Datenmigration bei Upgrades
 - KISS („Keep it small and simple“)
 - Skalierbarkeit (Funktionen, Schema, Performanz)

Lösung: Web-Application-Frameworks

- **Web-Application-Frameworks**

- Software, die die effiziente Entwicklung von Web-Applikationen unterstützt, vereinfacht und sicherer macht
 - Konzept: Die Mechanismen werden nur noch an den entscheidenden Punkten (Schemaentwurf, Applikationslogik, HTML-Design) angepasst
- Realisieren die o.g. Lösungsansätze (mehr oder weniger)
 - Datenbankabstraktion, Schemamanagement, Template-Engine
 - **Scaffolding** (typische Abläufe sind sehr leicht realisierbar)
- Es gibt eine große Zahl von ihnen, z.B.
 - **ASP.NET** C# / Visual Basic (Microsoft)
 - **AngularJS, NodeJS, Express** Javascript / Typescript (u.a. Google)
 - **JSF** Java / JavaBeans (SUN/Oracle, IBM)
 - **Zend Framework, Laravel, Symfony** PHP
 - **Django, Flask** Python
 - **Ruby on Rails** Rails
 - ...
 - siehe http://de.wikipedia.org/wiki/Liste_von_Webframeworks oder z.B. in einem [MDN Artikel](#)

Web-Application-Framework: Django

- **Django** ist ein auf der Sprache **Python** basierendes **Web-Application-Framework**
 - Integrierter **Objektrelationaler Mapper** (ORM)
 - Datenbank-Backend ist austauschbar (MySQL, Postgres, Oracle, SQLite, ...)
 - Schema-Definition und DB-Zugriff erfolgen High-Level, ohne Bedarf für SQL
 - Generiert zum Schema ein **komplettes Administrations-Interface**
 - Basierend auf Benutzerverwaltung mit fein steuerbaren Rechten
 - Mächtige Template-Sprache
 - Trennung von Datenaufbereitung und Darstellung (mit Vererbung zwischen Templates)
 - Flexibles URL-Management
 - Mustergesteuerte URL-Zerlegung und URL-Synthese
 - Sicher vor SQL-Injections, CSRF, Schutz vor XSS
 - ...

Programmiersprache Python

- **Python** Grundkonzepte

- universelle Multiparadigmen-Sprache
 - imperativ, objektorientiert, funktional
- dynamisch getypt (aber auch strikt getypt)
 - Typen sind an Daten / Objekte gebunden, nicht an Variablen
 - Ducktyping (→ [Wikipedia](#))
- Haupt-Ziele:
 - **Hohe Ausdrucksfähigkeit** (Probleme sind sehr kompakt lösbar)
 - **Sehr gute Lesbarkeit**
 - vielfältige Nutzbarkeit durch vielfältige Schnittstellen (Bibliotheken)
 - Erweiterbar durch Module
- Programme müssen nicht explizit übersetzt werden
- interaktiv nutzbar (Kommandos eingeben und direkt ausführen)
 - Python-Shell oder Erweiterung „ipython“

Programmiersprache Python

- Beispiele

Grundoperationen

```
[~] python3

>>> 1+2*3
7
>>> s = 'a' + "b"
>>> s
'ab'
>>> s == 'ab'
True
```

Strukturierte Anweisungen

```
>>> x = 3
>>> print(x)
3
>>> if x == 3:
...     print('x ist drei.')
... else:
...     print('x ist nicht drei.')
...
x ist drei.
```

Funktionen

```
>>> def max(a,b):
...     if a>b:
...         return a
...     else:
...         return b
...
>>> max(4,6)
6
```

Klassen, Methoden

```
>>> class C():
...     def f(self):
...         pass // do nothing
...     class D(C):
...         def f(self):
...             print("Ich bin D.")
...
>>> x = D()
>>> x.f()
Ich bin D.
```

Programmiersprache Python

- Beispiele für Datentypen und Ausdrücke

Tupel

```
>>> x = (0,1,2,3,4,5)
>>> x[3]
3
>>> x[-1]
5
>>> x[3:5]
(3, 4)
>>> x[3:4]
(3,)
>>> (3)
3
```

(3,) ist 1-Tupel
(3) ist Zahl

Listen

```
>>> x = [0,1,2,3,4,5]
>>> x[3]
3
>>> x.append('Y')
>>> x[3:]
[3, 4, 5, 'Y']
>>> x[:3]
[0, 1, 2]
>>> x[3:4]
[3]
```

Warum ist hier
[3,] unnötig?

Dictionaries

```
>>> s = dict(a=1, b=2)
>>> s
{'a':1, 'b':2}
>>> s == {'a':1, 'b':2}
True
>>> s['a']
1
>>> s.get('c', 'unknown')
'unknown'
>>> s.items()
[('a', 1), ('b', 2)]
```

```
>>> '%s + %s = %s' % (1,2,3)
'1 + 2 = 3'

>>> a, b = '', 'sonst'
>>> a or b, bool(a), bool(b)
('sonst', False, True)

>>> x = 'ja' if a else 'nein'
>>> print(x)
'nein'
```

```
>>> [ x*x for x in [1,2,3] ]
[1, 4, 9]

>>> p = [ (x, x*x) for x in [1,2,3] ]
>>> p
[(1, 1), (2, 4), (3, 9)]

>>> dict(p)
{1: 1, 2: 4, 3: 9}
```

„List-
Comprehension“

Programmiersprache Python

- **Python-Doku**



- Zentrale Python-Weseite: <https://www.python.org>
 - Enthält auch eine interaktive Shell: <https://www.python.org/shell/>
 - Doku: <https://docs.python.org/>
 - Interakt. Tutorials: <https://docs.python.org/3/tutorial/>
<https://www.learnpython.org/>
 - Weitere Tutorials: <https://wiki.python.org/moin/BeginnersGuide/Programmers>
- Wikipedia-Seite (Überblick):
[http://de.wikipedia.org/wiki/Python_\(Programmiersprache\)](http://de.wikipedia.org/wiki/Python_(Programmiersprache))
- Für frühere Python-Version (Python 2.x):
 - Python **2** Quick-Reference (PQR): <http://rgruet.free.fr>
 - **Sehr nützliches Nachschlagewerk**, leider bisher kein Python 3 Update
 - Aktuell PQR 2.7: <http://rgruet.free.fr/PQR27/PQR2.7.html>