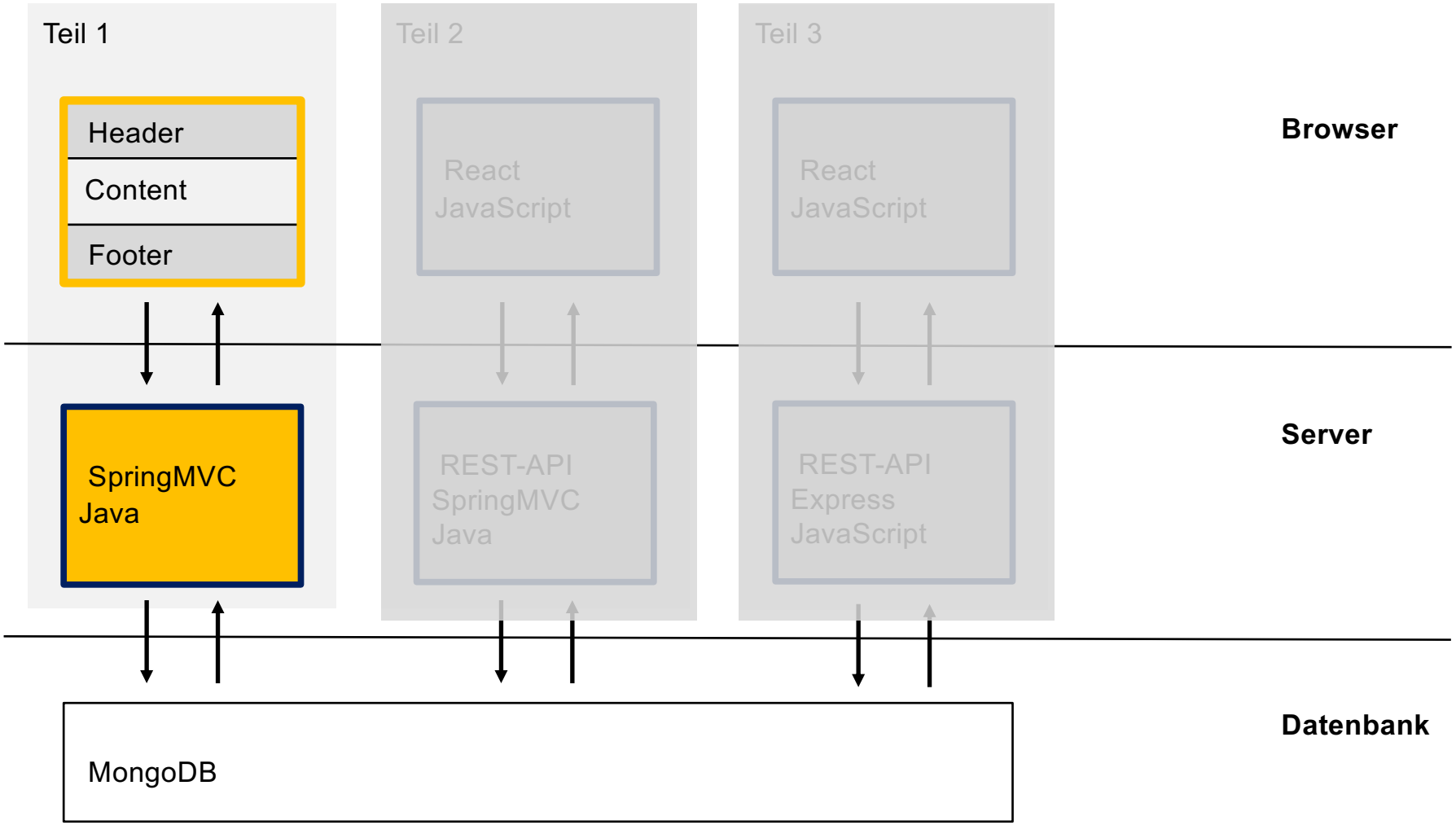
View Technologien

Themen heute

- Controller
  - Repetition Controller anhand UB3
- View Komponente im MVC Design Pattern
- View Design Pattern
  - Template View
  - Composite View
- View Technologien mit SpringMVC
  - Thymeleaf

Lab "flashcard": Setup



Besprechung Übung 3 (1/3)

```
@Controller
@RequestMapping("/hello")
public class HelloWorldController {
    @Autowired
    private QuestionnaireRepository questionnaireRepository;

    @GetMapping
    public @ResponseBody String sayHello(@RequestParam("name") String name) {
        List<Questionnaire> questionnaires = questionnaireRepository.findAll();
        String response = "Hello " + name + "<br/>"
                        + "You have " + questionnaires.size() + " Questionnaires in your repo.";
        return response;
    }
}
```

Besprechung Übung 3 (2/3)

- Zusammenfassung Dispatching
  - Das **Dispatching** wird vom **Front Controller** durchgeführt.
  - Im SpringMVC ist der Front Controller über das **DispatcherServlet** realisiert.
  - **Page Controller** realisieren die **fachliche Logik** der Applikation.
  - Der Front Controller leitet den Request aufgrund der URL an einen entsprechenden Page Controller weiter.
  - Ein Page Controller ist ein normales POJO und wird erst über die **Annotationen @Controller und @RequestMapping** zu einem Page Controller.

Besprechung Übung 3 (3/3)

- Zusammenfassung **Request Mapping**
  - Für das Mapping der Handler Methoden sind folgende Annotation wichtig:
    - **@RequestMapping** (siehe AB5)
      - **value** Mapping auf ein Path-Element
      - **method** Mapping auf eine HTTP Methode (oder **@GetMapping, @PostMapping,** ...)
      - **params** Mapping auf einen Request Parameter
      - headers, produces, consumes, ...
  - Zusätzlich kann mit folgenden Annotationen bei den Input Parameter Elemente aus der **Request-URL in der Methode** zugänglich gemacht werden:
    - **@PathVariable** (siehe AB5)
    - **@RequestParam** (siehe UB3)

Handling HTTP Request "/"

- Ein Request auf "/" kann folgendermassen behandelt werden
  - File "index.html" im Ordner "src/main/resources/public" mit redirect

```
<html>
<head>
    <meta http-equiv="refresh" content="0; url=questionnaires" />
</head>
</html>
```

  - oder mit "IndexController" und redirect

```
@Controller
@RequestMapping("/")
public class IndexController {
    @RequestMapping(method = RequestMethod.GET)
    public String index() {
        return "redirect:questionnaires";
    }
}
```

MVC Design Pattern

View

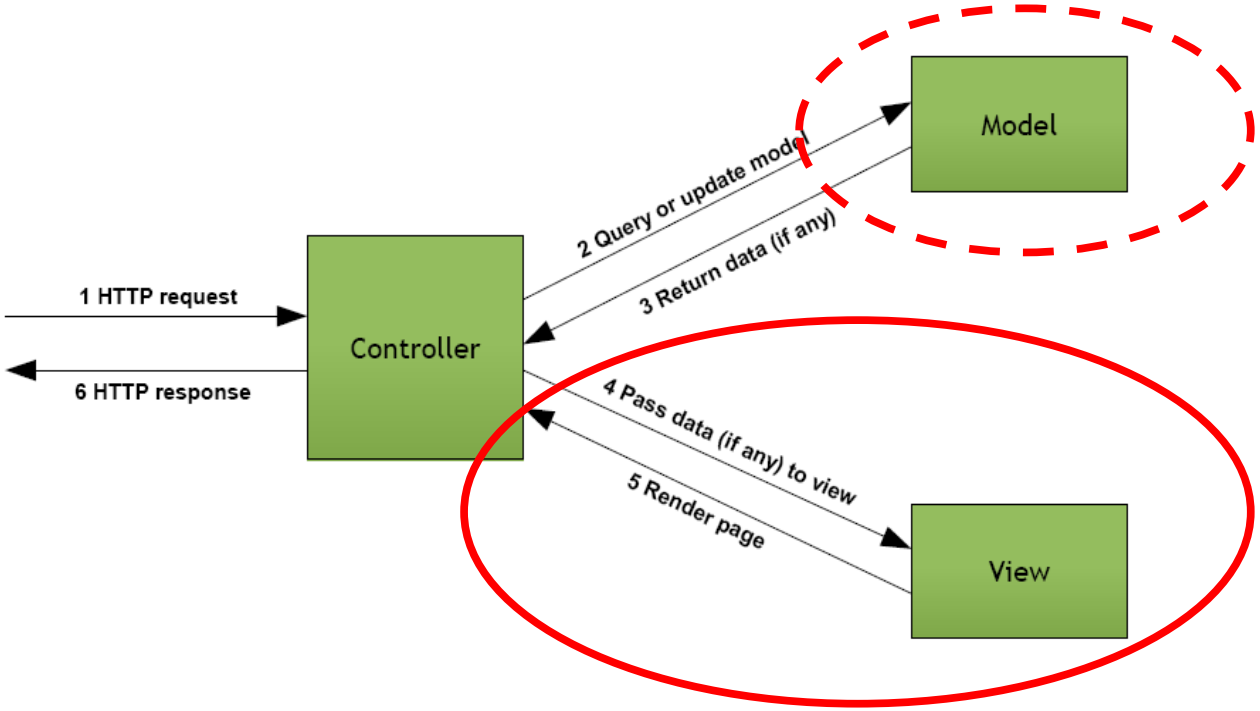
Displays the page using the model.

Model

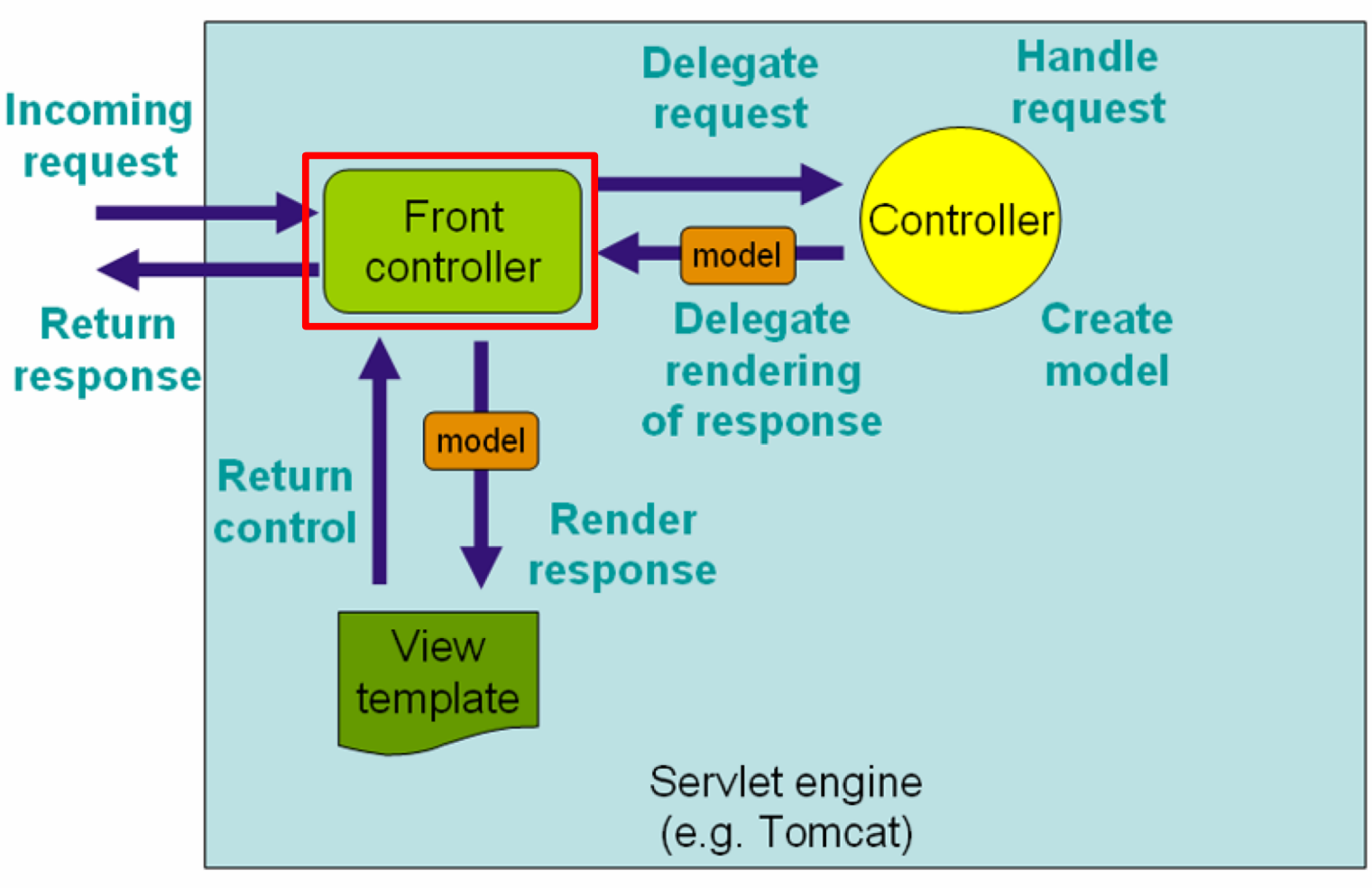
The data required for the request.

Controller

Handles the request, generates the model and decides on the view

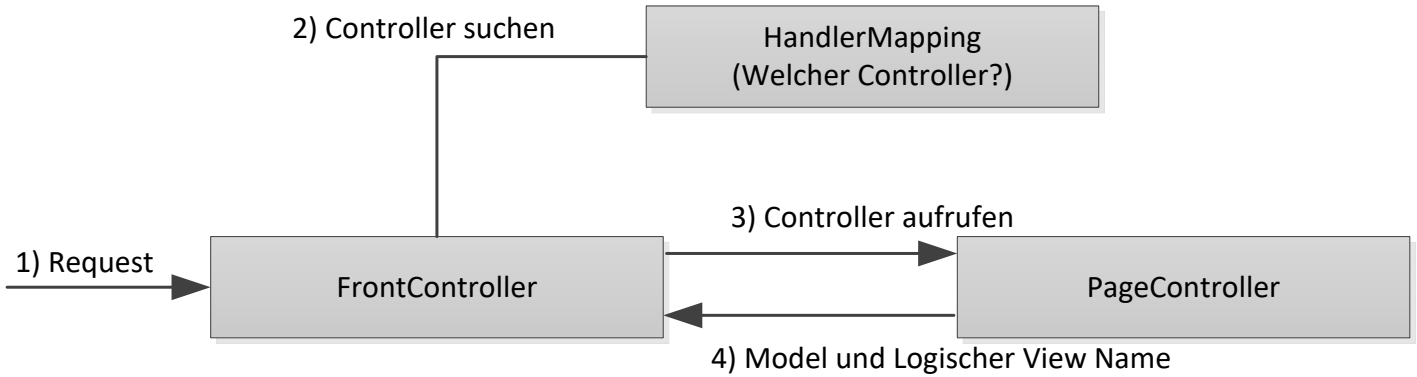


SpringMVC Workflow



see Spring Reference Documentation "22.2 The DispatcherServlet"

Page Controller Response



- Response
  - **Model**
  - **Logischer View Name**

Model

- Gemäss Spring Reference Documentation
  - "The model (the M in MVC) is a **java.util.Map** interface, which allows for the complete abstraction of the view technology."

- Spring abstrahiert die Map durch das Interface "org.springframework.ui.Model"
  - **Das Model Interface kann in einer Controller Methode als Input Parameter injiziert werden.**

    Kommt vom Spring framework "automatgisch". Referenz nicht verändern!

    public String findAll(**Model model**) { ...

View Design Pattern

- Template View Pattern
  - mit Thymeleaf
- Composite View Pattern
  - mit Thymeleaf

Thymeleaf als View Technologie

- XHTML/HTML5 Template Engine die vollständig im Java Spring MVC integriert ist
- Unterstützt "Natural Templating"
- Unterstützt Layout- und Decorator-Architektur

- siehe http://www.thymeleaf.org/

Arbeitsblatt 7: Template View Pattern

- HTML Code aus dem Controller entfernen
  - View Template erstellen
    - Namespace für Thymeleaf setzen
    - HTML Page erstellen
      - Dynamische Elemente durch Thymeleaf Expressions ausdrücken
  - Controller anpassen
    - View Name im RETURN Wert setzen
    - Spring "Model" Interface als Modell für die View nutzen
- **Demo mit findAll()**

Pages in einer Hierarchie

- Hierarchie in einer Parent - Child Beziehung
  - Parent → Layout
  - Child → Subviews
- Im Layout wird die Webpage aus Subviews zusammengesetzt:
  - Include Style Layout
    - Jede View schliesst die Subview mit ein
    - Einfach
    - Für grosse Applikation ungeeignet, da jede View bei Änderungen angepasst werden muss.
  - Hierarchical Style Layout
    - Modulares Design mit Layout/Decorator und Fragments
    - Komplex, aber sehr sauber Strukturierung auch in grossen Applikationen möglich, da das Layout genau in einer Page definiert wird.

Arbeitsblatt 8: Composite View Pattern

- Redundanz aus HTML Pages entfernen
  - Layout Template erstellen
  - Platzhalter für Content Bereich
    - Footer über Include Style Layout
      - Footer / Header
    - Content über Hierarchical Style Layout
      - Fragments in der Subviews

- **Demo für Liste der Questionnaires**
  - **Subview "footer.html"**
  - **Subview "list.html"**

Übung 4: Create Formular implementieren

- Hausaufgabe

- CREATE (**C**RUD) umsetzen: **2 Schritte notwendig!**

  1. **GET Request**
     Create Formular beim Server holen

  2. **POST Request**
     Neuer Questionnaire über POST Request erzeugen

- Applikation ausbauen