Einführung in React

Info zur Prüfung

Datum	**29.10.17, 15:15-16:45**
Raum	**3.-111**
Dauer	90 Minuten
Stoff	bis und mit Lektion05
Ablauf	**Teil 1 (30 Minuten)**
	keine Unterlagen
	Teil 2 (60 Minuten)
	schriftl. Unterlagen erlaubt
	keine elektr. Hilfsmittel
	Schreibpapier mitnehmen

Themen heute

- Besprechung Übung 5
- Klassische Webapplikationen vs. Single-Page Applikationen
- Einführung in React
  - Komponentenarchitektur
  - Properties und State einer Komponente
  - Virtueller DOM

Besprechung Übung 5 (1/5)

- Die Subview ist durch das File "update.html" im Ordner "src/main/resources/templates/questionnaires" implementiert.
- Die zentralen Elemente in der Subview sind:
  - Das Model-Objekt hinter dem Formular
    - Eine Instanz von Questionnaire
  - Der Zugriff auf das Model-Objekt
    - Über den Key "questionnaire" in einer "Model"-Instanz
    - Thymeleaf Expression th:object="${questionnaire}"
  - Der Zugriff auf Properties dieses Model-Objektes, wie "title
    - Über die Thymeleaf Funktion th:field="$*{title}"
  - HTTP Request, der aus dem Formular ausgelöst werden soll
    - HTTP-PUT auf th:action="@{/questionnaires}"

Besprechung Übung 5 (2/5)

```
<form action="#"  th:action="@{/questionnaires} +'/'+${questionnaire.id}"
                  th:object="${questionnaire}"
                  method="post" >
   <input type="hidden" name="_method" value="PUT" />
   <input type="text" th:field="*{title}"/>
   <input type="submit" value="New"/>
</form>
```

`th:action="@{/questionnaires}+'/'+${questionnaire.id}"`

Legt für das Formular die Aktion als Request URL fest. Notation **@{...}** ist wichtig.

`th:object="${questionnaire}"`

Referenziert das Model-Objekt über den Key "questionnaire"

`th:field="*{title}"`

Referenziert die Property "title" des Model-Objekts. Notation ***{...}** ist wichtig.
Das Model-Objekt muss eine entsprechende Setter-Methode zur Verfügung stellen.

Besprechung Übung 5 (3/5)

`th:method="put"`

Unterstützung von Thymeleaf, um "hidden" Input nicht schreiben zu müssen

```
<form action="#"  th:action="@{/questionnaires} }+'/'+${questionnaire.id}"
                  th:object="${questionnaire"
                  th:method="put" >
   <!-- <input type="hidden" name="_method" value="PUT" /> -->
   <input type="text" th:field="*{title}"/>
   <input type="submit" value="New"/>
</form>
```

Besprechung Übung 5 (4/5)

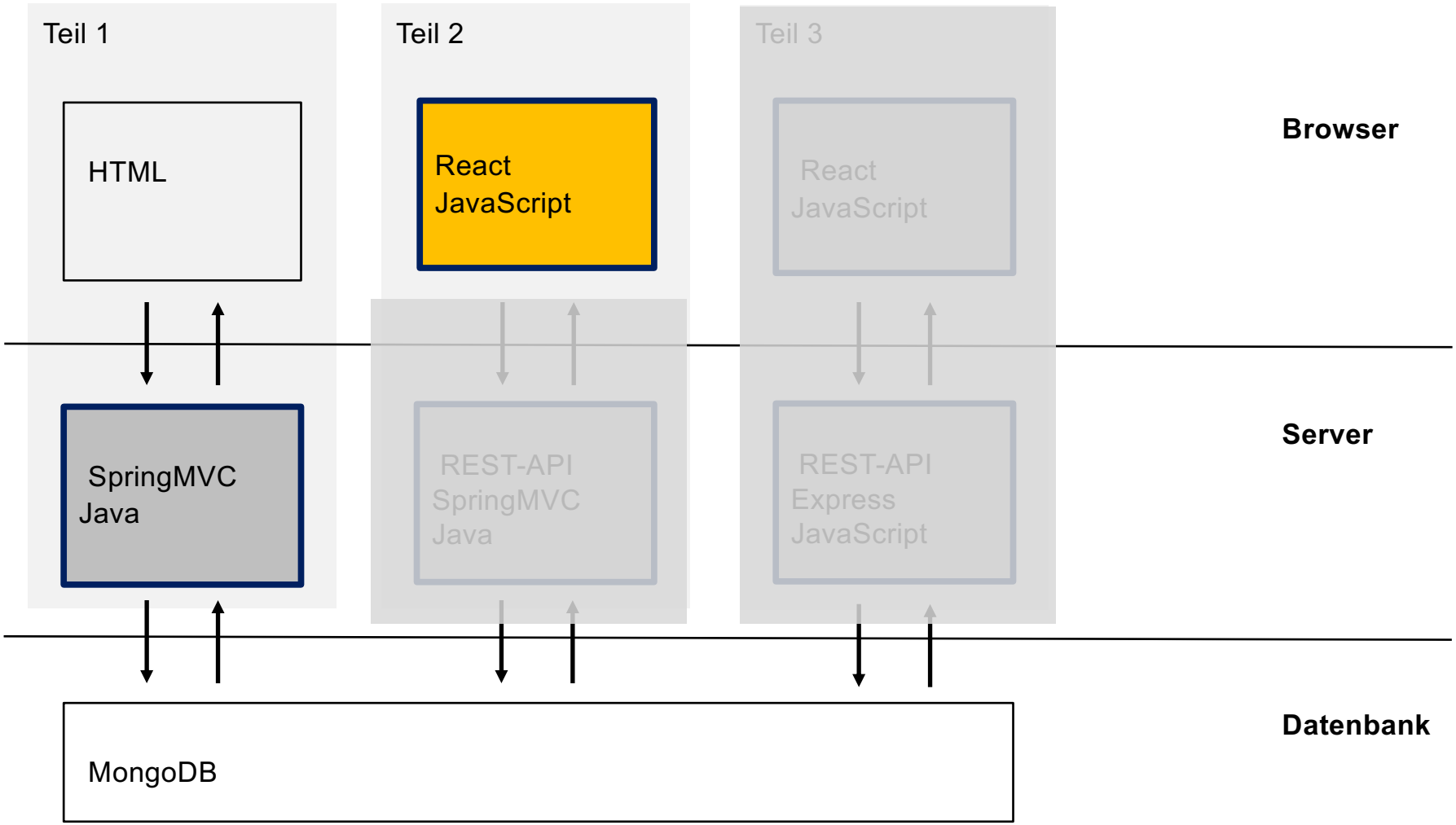
QuestionnaireController mit UPDATE Funktionalität inkl. Error Handling:

```
@GetMapping(value = "/{id}", params = "form")
public String updateForm(@PathVariable String id, Model uiModel) {
      Optional<Questionnaire> questionnaireOptional =
                                    questionnaireRepository.findById(id);
      if (questionnaireOptional.isPresent()) {
            Questionnaire questionnaire = questionnaireOptional.get();
      uiModel.addAttribute("questionnaire", questionnaire);
      logger.debug("Successfully delivered update form");
      return "questionnaires/update";
   } else {
      logger.info("No entity found with id=" + id);
      return "404";
   }
}
```

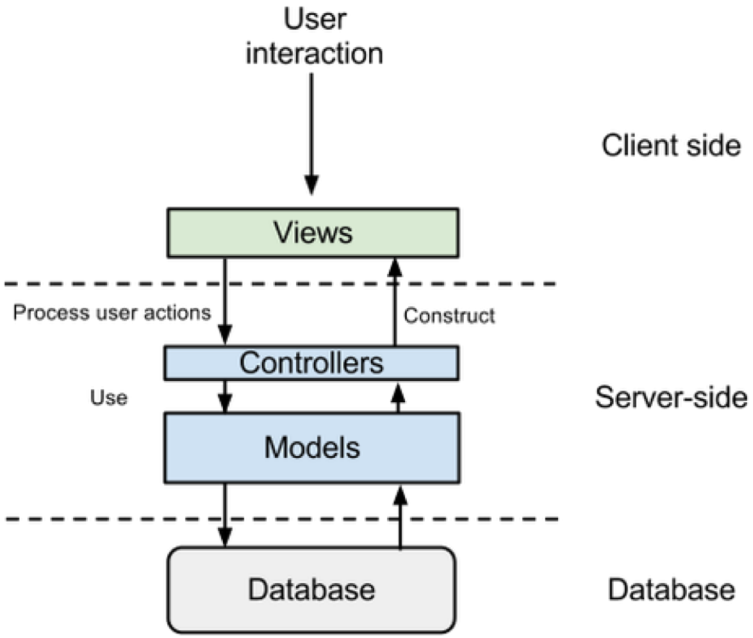
Besprechung Übung 5 (5/5)

```
@PutMapping(value = "/{id}")
public String update(@PathVariable String id,
      @Valid Questionnaire questionnaire, BindingResult bindingResult) {
   if (bindingResult.hasErrors()) {
      logger.debug("Binding error: " + bindingResult.getAllErrors());
      return "questionnaires/update";
   }
   Optional<Questionnaire> questionnaireOptional =
                                    questionnaireRepository.findById(id);
   if (questionnaireOptional.isPresent()) {
      Questionnaire oldQuestionnsaire = questionnaireOptional.get();
      oldQuestionnsaire.setDescription(questionnaire.getDescription());
      oldQuestionnsaire.setTitle(questionnaire.getTitle());
      questionnaireRepository.save(oldQuestionnsaire);
      logger.debug("Successfully updated questionnaire " + id);
   }
   return "redirect:/questionnaires";
}
```

Lab "flashcard": Setup



Klassische Webapplikation: Architektur- und Interaktionsmodell

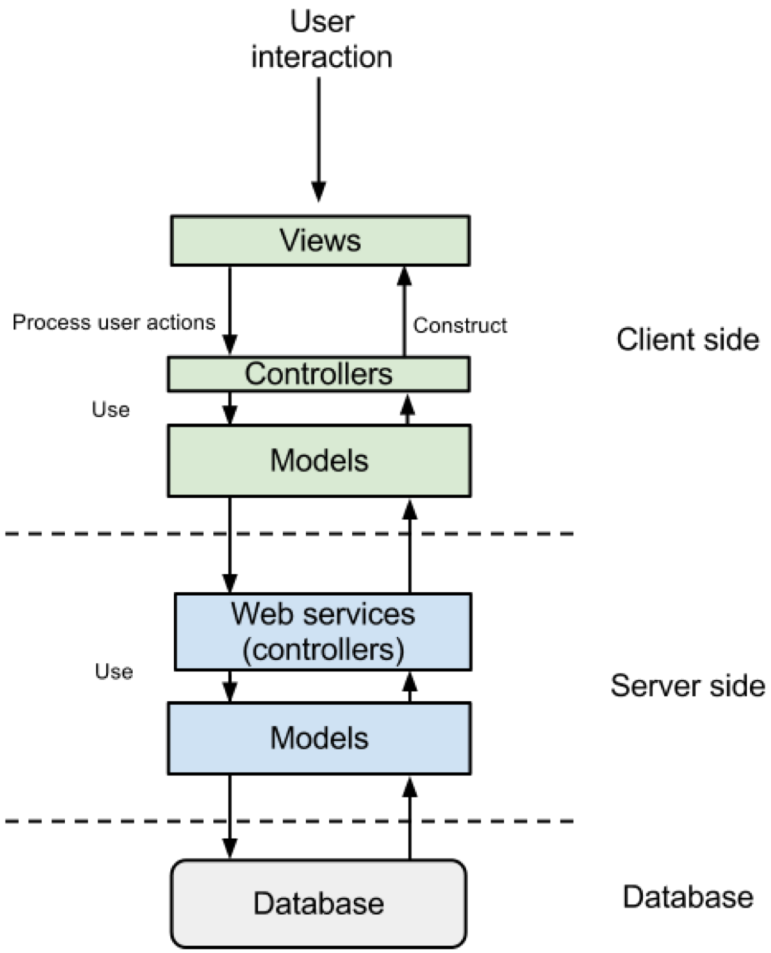


Vorteile	Nachteile
Full Control	User Experience
Sicherheit	Server-Roundtrip
...	...

Was ist eine Single Page Application (SPA)?

- Eine Single Page Application ist eine Webanwendung, die **keinen Seitenwechsel** (Roundtrip) durchführt, sondern die Anzeige nur durch Austausch von Seitenelementen via JavaScript/DOM verändert. Es gibt dabei also keine serverseitige Seitennavigation. Die URL ändert sich grundsätzlich nicht (kann aber simuliert werden!).
- **Initial wird eine komplette Seite** oder zumindest das Grundgerüst einer Webseite vom dem Server geladen. Die Seite lädt anschliessend **Daten über Webservices** (meist REST-basierte Dienste) nach und erzeugt die Darstellung clientseitig (clientseitiges Rendern).
- Eine SPA **wirkt damit wie eine Desktopanwendung**.

SPA: Architektur- und Interaktionsmodell



Vorteile	Nachteile
User Feedback	Browser-Unterstützung
Client- vs. Server-Logik	Verschiedene Technologien
...	...

Clientseitige Technologien

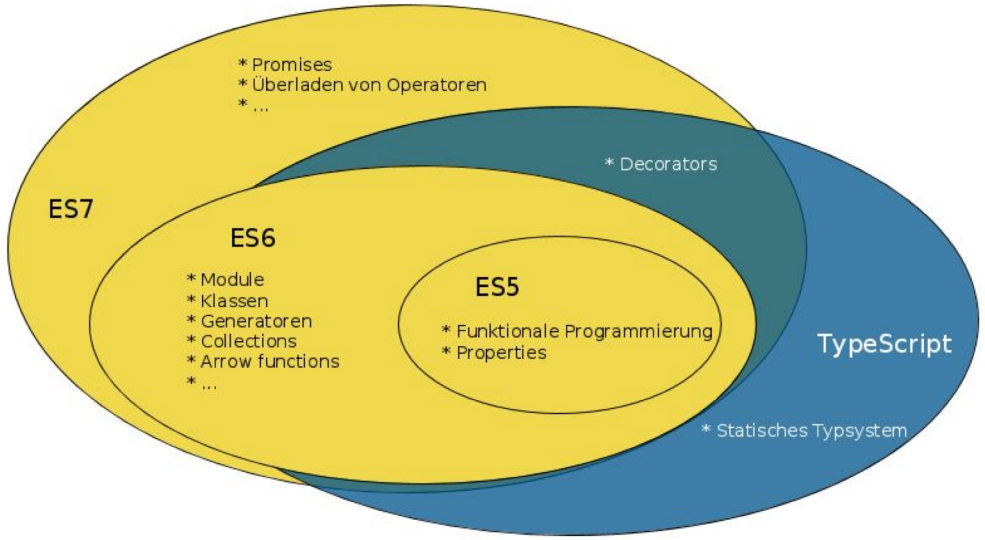
- Eigene Runtime Umgebung
  - Adobe Flex → Apache Flex
    - ursprünglich von Adobe, jetzt Open Source bei Apache
    - FlashPlayer notwendig
  - Java Applets und JavaFX (via Web Start)
    - JRE oder JavaFS Runtime Umgebung notwendig
  - Microsoft Silverlight
    - Runtime Umgebung notwendig
- Skriptsprachen
  - Visual Basic Script (VBScript)
    - Microsoft
    - IE Support!
  - ActionScript
    - Adobe
  - **ECMAScript (JavaScript Standard)**
    - standardisierter Sprachkern basierend auf JavaScript
    - sehr breite Unterstützung in allen wichtigen Browser, auch auf mobilen Geräten
    - aktuelle Version: ECMAScript 6 (oder ECMAScript 2015), seit Juni 2015

JavaScript ist nicht Java!

- **Kein Typsystem**: Es kann keine Typinformation mit Variablen verknüpft werden. D.h. Variablen sind Behälter in die alles abgefüllt werden kann und es liegt in der Verantwortung des Programmierers sicherzustellen, dass die richtigen Daten bei deren Verwendung auch vorliegen.
- **Interpretiert**: Zusammen mit dem fehlenden Typsystem heisst dies für den Entwickler in erster Linie: Viele Fehler manifestieren sich erst zur Laufzeit. Es existiert kein Compiler der frühzeitig auf Probleme aufmerksam macht.
- **Keine Klassen und damit auch keine Vererbung**: Will man Vererbungshierarchien mit JavaScript verwenden, müssen diese von Hand ausprogrammiert werden. Dies ist möglich aber umständlich.
- **Objektorientiert**: Ein Objekt ist im Wesentlichen eine Hashtabelle an die alles Mögliche angehängt werden kann, z.B. Variablen, Funktionen und Arrays. Da keine Klassen und kein Typsystem existieren, können theoretisch alle Objekte unterschiedlich sein.
- **Funktionen sind auch Objekte**! Es ist möglich Funktionen als Parameter zu übergeben oder als Rückgabewerte zu erhalten. Dies ist eine grosse Stärke von JavaScript gegenüber Java.

ECMAScript 6: Neue Features

- Übersicht
  - "ECMAScript 6 — New Features: Overview & Comparison" http://es6-features.org/
- Zusammenfassung
  - Constants
  - Arrow Functions
  - Classes
  - Modules
  - ...



- **ABER**: Es unterstützen noch nicht alle Browser ECMAScript 6 => Übersetzung des Quellcodes auf ECMAScript 5 mit Transpiler wie **Babel** wird notwendig!

AB12: HelloWorld mit React und ECMAScript 6

- Entwicklungsumgebung aufsetzen
  - Text Editor
    - Atom, Sublime, **Visual Studio Code**
  - Package Manager
    - **npm**
      um JavaScript Packages aus einem zentralen Repository lokal zu installieren (Directory node_modules)
  - Package "create-react-app"
    - enthält **webpack, babel, ...**
      um Frontend Assets (*.css, *.js, ...) zusammenzufügen und daraus ein einziges File zu generieren => File "bundle.js"
    - enthält Dev-WebServer mit Live Reload
  - Browser Tools
    - React Plugin für Chrome, Firefox, ...

Was ist React?

- React ist eine JavaScript-Bibliothek zum Erstellen von Benutzeroberflächen.
- React wurde 2013 von Facebook veröffentlicht.
- React ist Open Source.
- React ist als Bibliothek konzipiert.
- React ist lediglich für die View in MV* Kontext zuständig.
- React besteht im Wesentlichen aus folgenden Kernelementen:
  - **Komponentenarchitektur**
  - **Virtueller DOM**
  - Browserkompatibilität über synthetische Events, um jQuery-artig Browserunterschiede wegzuabstrahieren

Komponentenarchitektur

- Komponenten sind die **zentralen und einzigen Bausteine** einer React-Applikation.
  - **React-Komponente "App"**
    - **App.js**
- Einbindung der React-Komponente "App" in die SPA
  - **index.html**
    - importiert React Runtime
    - stellt HTML-Root-Element bereit
  - **index.js**
    - importiert Komponente App
    - bindet an HTML-Root-Element

Die Properties einer Komponente

- Properties werden von aussen an die Komponente übergeben, analog Input Parameter einer Methode
- Sie sind innerhalb der Komponente nicht veränderbar.

```
import React, { Component } from 'react';

class App extends Component {
  render() {
    return (
      <h1>{this.props.message}</h1>
    );
  }
}

export default App;
```

Properties 'message' wird ausgelesen

```
ReactDOM.render(
  <App message="Hello from index.js"/>,
  document.getElementById('root'));
```

Properties 'message' wird übergeben

Der State einer Komponente (1/2)

- Eine React-Komponente kann einen Zustand (= State) besitzen, der sich zur Laufzeit ändert. Der State beschreibt die **veränderlichen Daten** der Komponente.

- **Wichtig**: Nur die Komponente selber kann seinen State mit setState(...) verändern.

- Ein Änderung im State mit **setState(...)** löst **automatisch** einen Call auf die Methode "render()" aus
  → **Komponente wird neu gezeichnet.**

Der State einer Komponente (2/2)

```
import React, { Component } from 'react';

class App extends Component {
    constructor(props) {
        super(props);
        this.names = [
            'Balin', 'Dwalin', 'Fili', 'Kili', 'Dori', 'Nori', 'Ori', 'Oin', 'Gloin',
            'Bifur', 'Bofur', 'Bombur'
        ];
        this.state = {
            index: 0
        };
        this.tick = this.tick.bind(this);
        setInterval(this.tick, 1000);
    }

    tick() {
        const newIndex = (this.state.index + 1) % this.names.length;
        this.setState({index: newIndex});
    }

    render () {
        return (<h1>{this.props.message} {this.names[this.state.index]}</h1>)
    }
}

export default App;
```

Der Call "super(props)" ist notwendig!

'index' wird als State definiert

Die Methode 'tick' wird jede Sekunde aufgerufen. Mit dem Binding wird die Methode 'tick' an die Instanz gebunden. **Wichtig!**

In 'tick' wird der State 'index' neu berechnet
=> Komponente wird neu gezeichnet
=> **Wichtig**: Muss über **setState()** erfolgen!

Lesender Zugriff auf State 'index'

Der Lifecyle einer Komponente

```
import React, { Component } from 'react';

class App extends Component {
    constructor(props) {
        super(props);
        this.names = [
            'Balin', 'Dwalin', 'Fili', 'Kili', 'Dori', 'Nori', 'Ori', 'Oin', 'Gloin',
            'Bifur', 'Bofur', 'Bombur'
        ];
        this.state = {
            index: 0
        };
        this.tick = this.tick.bind(this);
    }
    componentDidMount() {
        this.timer = setInterval(this.tick, 1000);
    }
    componentWillUnmount() {
        clearInterval(this.timer);
    }
    tick() {
        this.setState({index: (this.state.index + 1) % this.names.length});
    }
    render () {
        return (<h1>{this.props.message} {this.names[this.state.index]}</h1>)
    }
}
export default App;
```

Die Methode "componentDidMount()" wird von React aufgerufen, sobald die Komponente an virtuellen DOM-Tree eingebunden ist.

Die Methode "componentWillUnmount()" wird aufgrufen kurz bevor die Komponente aus dem DOM-Tree entfernt und gelöscht wird.

AB13: State & Properties

- Erweiterung der Komponente "HelloWorld"
  - um State "index"
  - um Property "message"

  - analog Vorlesungsskript

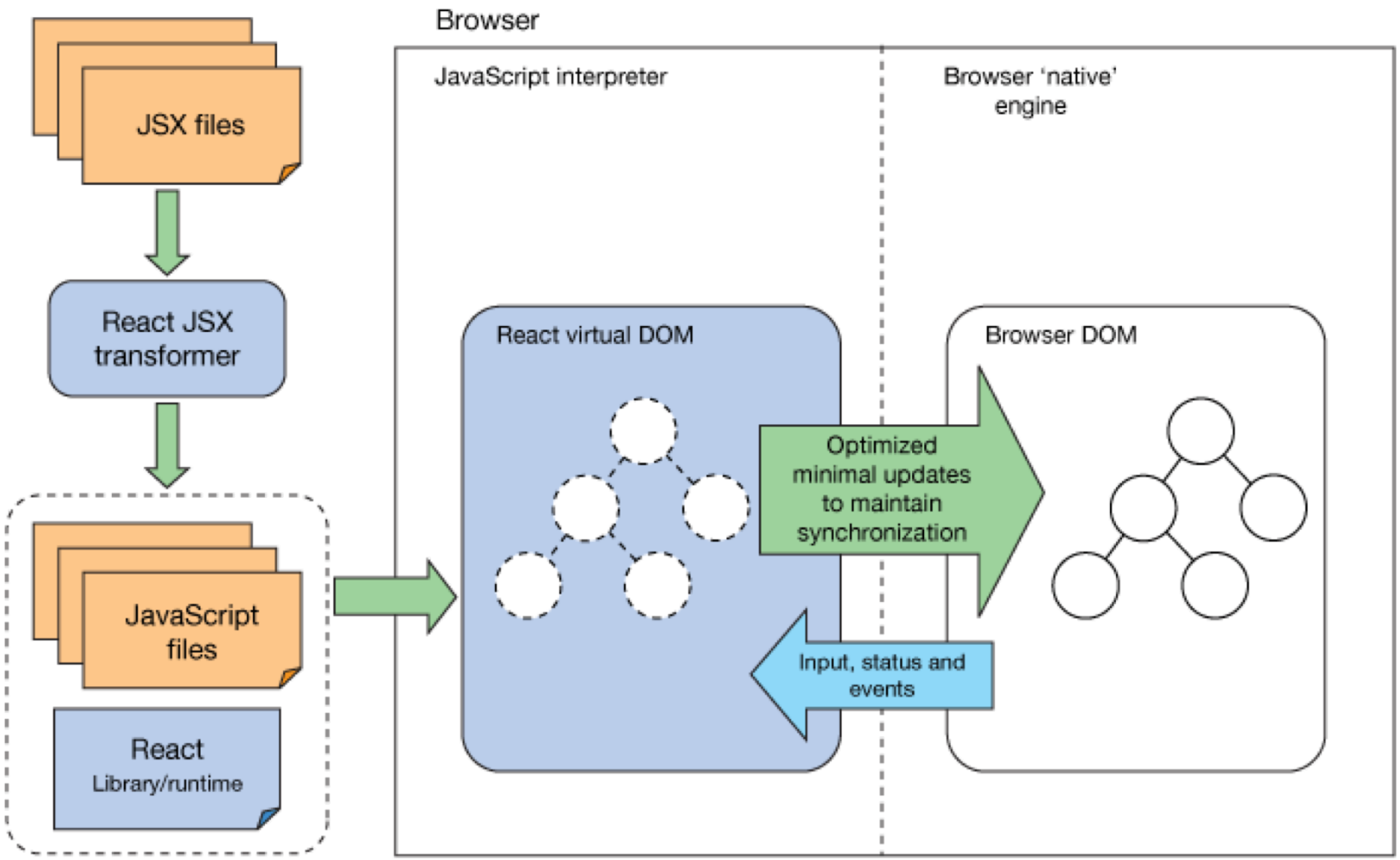
Virtueller DOM (1/3)

- Die Manipulation des DOM (Document Object Model) im Browser ist teuer - und deshalb langsam
  - Jede Änderung an dieser Baumstruktur quittiert der Browser mit teurer Neuberechnung seiner Geometrie.
  - Je mehr geändert wird, desto länger dauert es. Je weniger man den DOM des Browsers verändert, desto schneller ist die Applikation.
- JavaScript selber ist performant
  - Viele JavaScript-Frameworks suchen einen Kompromiss mittels Verwendung von altbewährten Entwurfsmustern, um die Komplexität zwischen Einfachheit und Performance zu bändigen (Two-way binding, dirty-checking, … etc.).
  - React versucht es mit einem extremen Ansatz, der in erster Linie die Einfachheit und nicht die Performanz in den Fokus stellt:
    React rendert bei jedem Update einfach alles neu.

Virtueller DOM (2/3)

- Virtueller DOM
  - Mit React Komponenten und JSX arbeitet man nicht direkt mit dem DOM des Browsers, sondern mit normalen JavaScript Objekten (=> Virtueller DOM), die schnell gelesen und bearbeitet werden können, ohne dass damit tatsächliche Änderungen am DOM ausgelöst werden.
  - Bei jeder Änderung der Daten erstellt React einen neuen virtuellen DOM.
  - Ein stark optimierter und heuristischer Algorithmus vergleicht diesen neuen Baum mit den vorherigen und errechnet eine Liste von minimalen Änderungen am richtigen DOM aus.
  - Diese werden gesammelt und nicht direkt, sondern im Batch an den Browser weitergeleitet.

Virtueller DOM (3/3)



from https://www.ibm.com/developerworks/library/wa-react-intro/

Übung 6 als Hausaufgabe

- App "react-counter" implementieren