FS18

WEB ENGINEERING & DESIGN 3

Introduction

T. Bocek

thomas.bocek@hsr.ch

Slides partially by Silvan Gehrig

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

Learning Objectives

The learning objective of this lecture is to acquire knowledge and expertise in single page applications (SPA) and to be able to understand its implementation.

The students are able to...

- **… differentiate between a traditional web application and an SPA**
- **… explain key concepts of an SPA**
- **… to write a simple SPA**
- **… to explain the need for web frameworks such as Angular, React, or Vue.js**

Topics

- **In the News**
- **Introduction**
  - Traditional Web vs. SPA
- **Build a Non-SPA application with golang**
- **Build an SPA with golang**

URL: b.socrative.com/student/

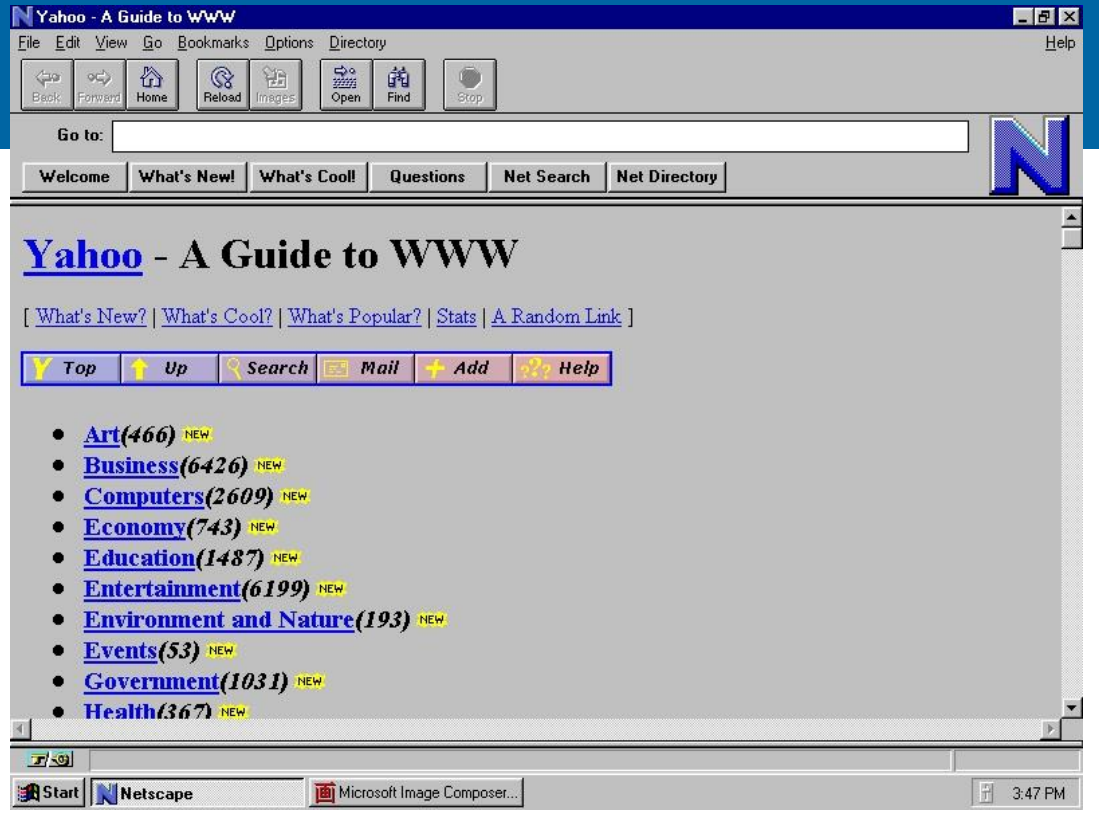
Room: HSR

Single Page Application (SPA) - In the News

- **29.01.2019: You probably don't need a single-page application**
  - Devs today: pre-made choice: REST API on the backend, and a React/Angular/Vue/Elm frontend
  - No special requirements: traditional server-rendered architecture faster
    - Stateless requests
      - One request, you have the data vs. SPA many requests → synchronization problems
    - The browser knows how to deal with a traditional architecture
      - Manage history
    - Fewer, more mature tools
      - learned Gulp, CoffeeScript, BackboneJS, and SASS
    - SEO for free
      - ~~can be indexed by crawlers~~
- **Complexity = bad**
  - Software bugs and development slowdown
- **When SPA**
  - Core functionality is real-time (e.g. Slack)
  - Rich UI interactions are core to the product (e.g Trello)
- **Hybrid**
  - Don't need to use the SPA paradigm for your whole app
- **21.02.2019: Faktencheck zu Progressive Web Apps, Teil 4: Grenzen & Auswege**
  - Können Progressive Web Apps alles, was native Anwendungen auch können?
    - Nein.

Introduction – Preface

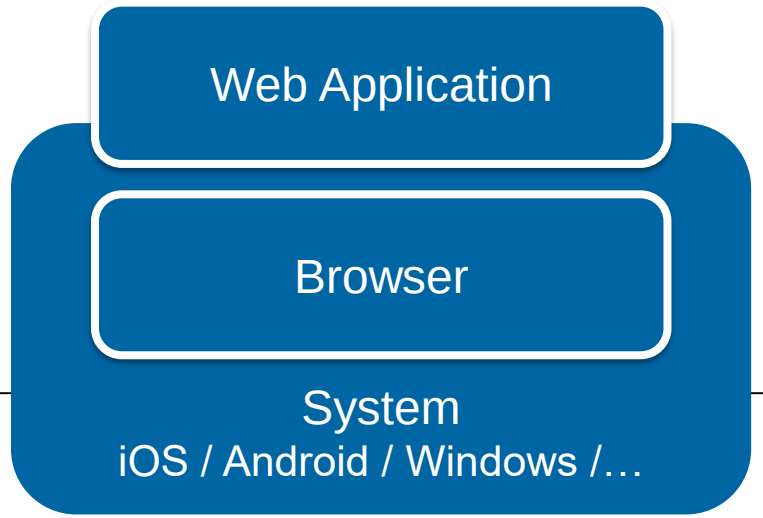
- **Browsers are getting more powerful (history)**
  - Nearly every device accommodates a browser
  - …more and more hardware access
    - Camera
    - Geo-Information
    - …and even more https://developer.mozilla.org/en-US/docs/WebAPI
    - https://caniuse.com



https://www.w3.org/2010/Talks/0921-html5-plh/web10.html (download)

Today, browsers provide the required feature-set

and performance to run sophisticated, web-based applications



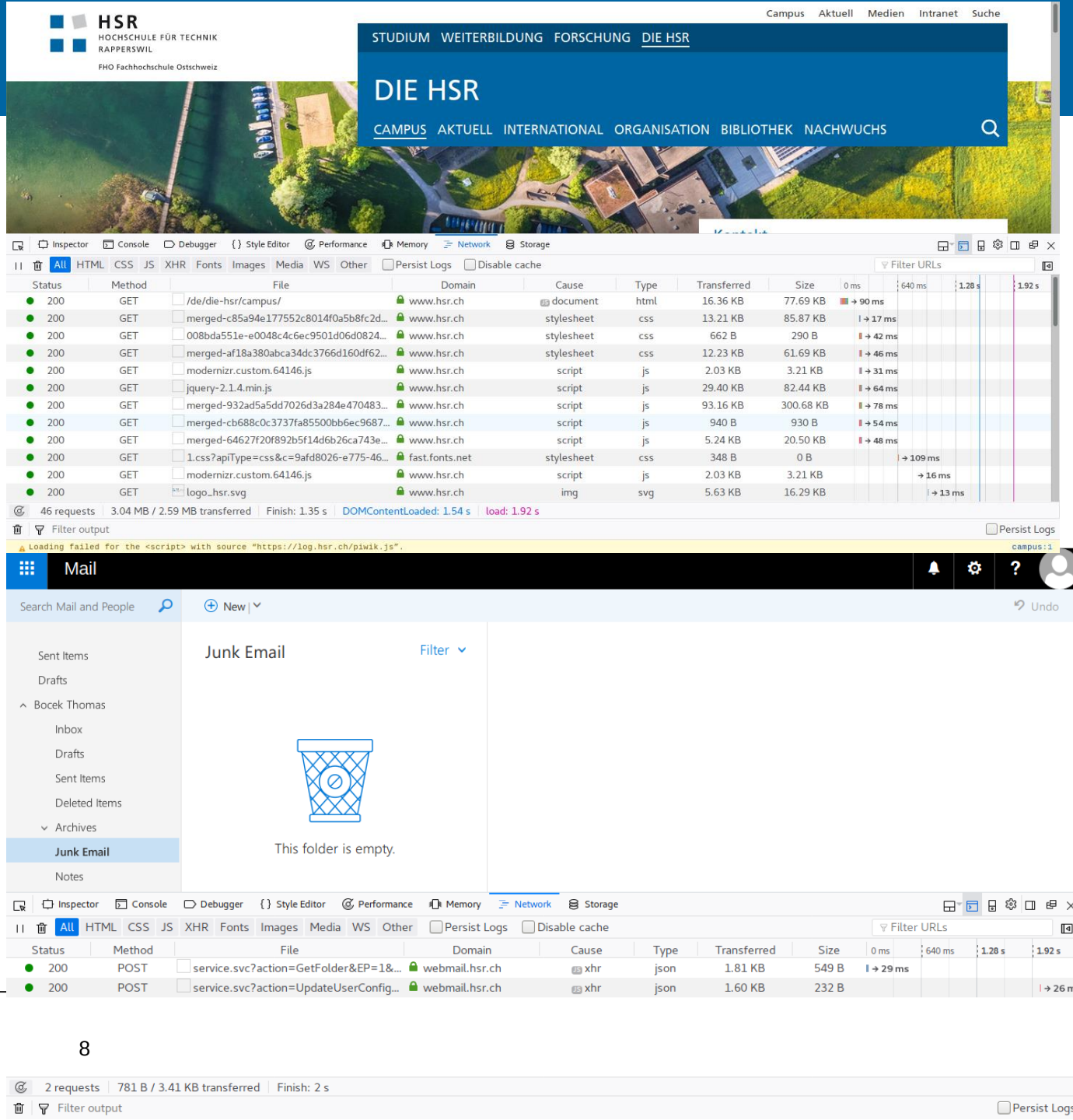
Introduction – Browser-based Applications

- **Benefits**
  - Work from anywhere anytime
    - Can be accessed from any device connected to the Internet. Data is available from anywhere (user logs in). Productivity advantage.
  - Platform independent, including mobile
  - No software update, no app installation, easy maintenance.
  - Software can be provided as a Services (SaaS)
    - «pay as you go» (charges based on usage)
  - Development: huge community
  - Can be used offline: electron.io
  - Standard-based (W3C)

- **Liabilities**
  - No data sovereignty
  - Limited/restricted hardware access
  - No operation system access
  - If Internet connectivity slow, application will be slow. If network is down, application is also down
  - Security/privacy
  - Applications can be slower than native desktop apps. When using electron.io you are bundling the whole web browser
  - Stuck with JavaScript (however, its evolving, WebAssembly)

Single Page Application (SPA)

- **Website**
  - No need to load entire page from server
  - Interacts with user by rewriting parts of the DOM
  - Example non-SPA: https://dsl.hsr.ch
  - Example SPA: https://webmail.hsr.ch
- **SPA moves logic from the server to the client**
  - Server typically data API service
  - "Thin Server Architecture“
- **Lack of JavaScript execution on crawlers**
  - Content may not be indexed – email no issue, documentation pages big issue

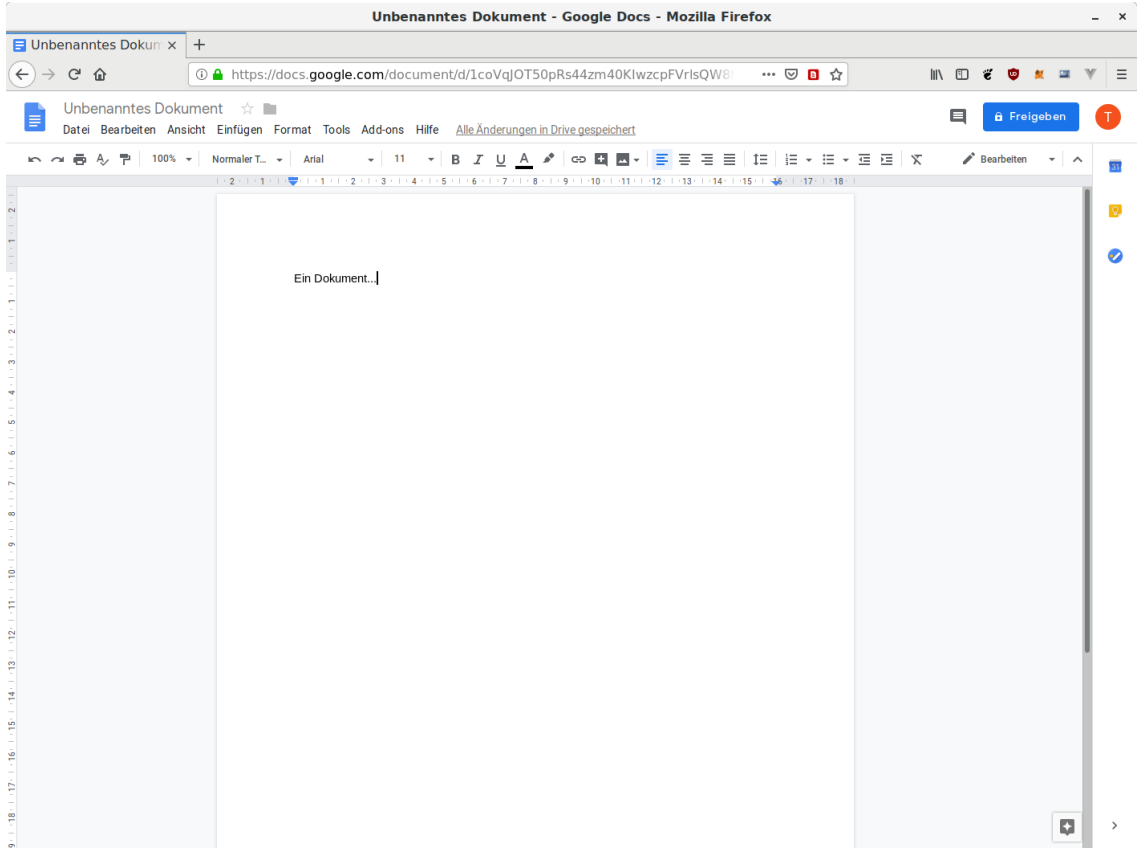


URL: b.socrative.com/student/

Room: HSR

Single Page Application (SPA)

- **Google Docs / Sheet / Slides**
  - Word processor in the browser
  - Launched as Writely by Upstartle in Aug 2005
  - Bought by Google in 2006
  - In 2009 removed Beta status
  - 2010 DocVerse, mulit user collaboration
  - 2012 Quickoffice, file formats
- **Google Sheets most likely used for assigning Mini-project**



Introduction – SPA Definition

- **A Single Page Application (SPA)**

…is a web application or web site that interacts with the user by dynamically rewriting the current page rather than loading entire new pages from a server.

In an SPA, either all necessary code – HTML, JavaScript, and CSS – is retrieved with a single page load, or the appropriate resources are dynamically loaded and added to the page as necessary, …

…providing a user experience similar to that of a desktop application.

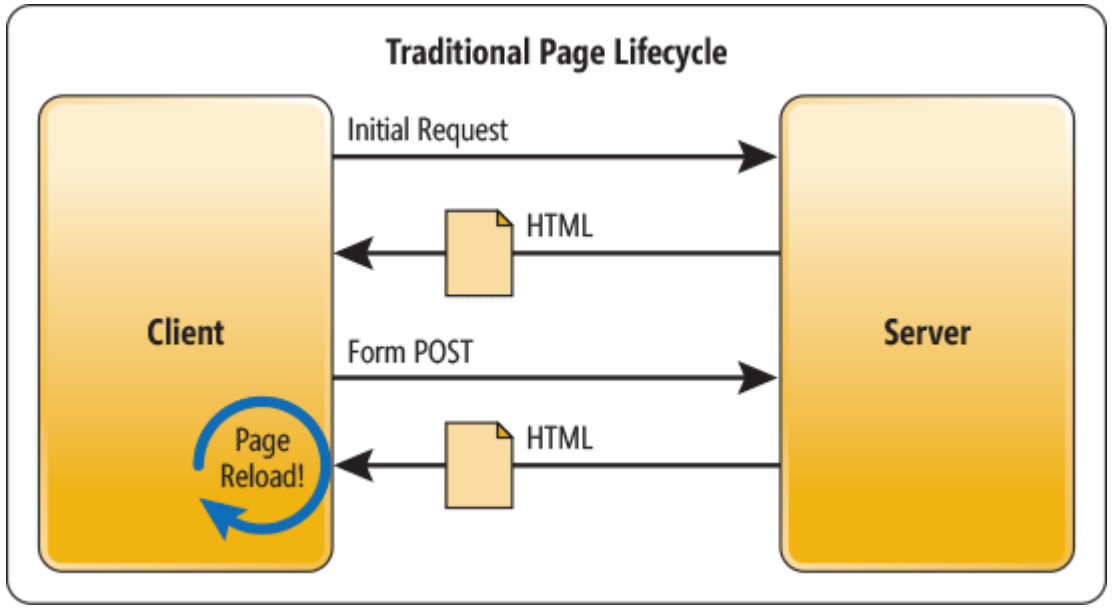
- ***SPAs use AJAX and HTML5 to create responsive Web apps, without constant page reloads.***

Source: https://en.wikipedia.org/wiki/Single-page_application

SPA vs. Traditional / MPA

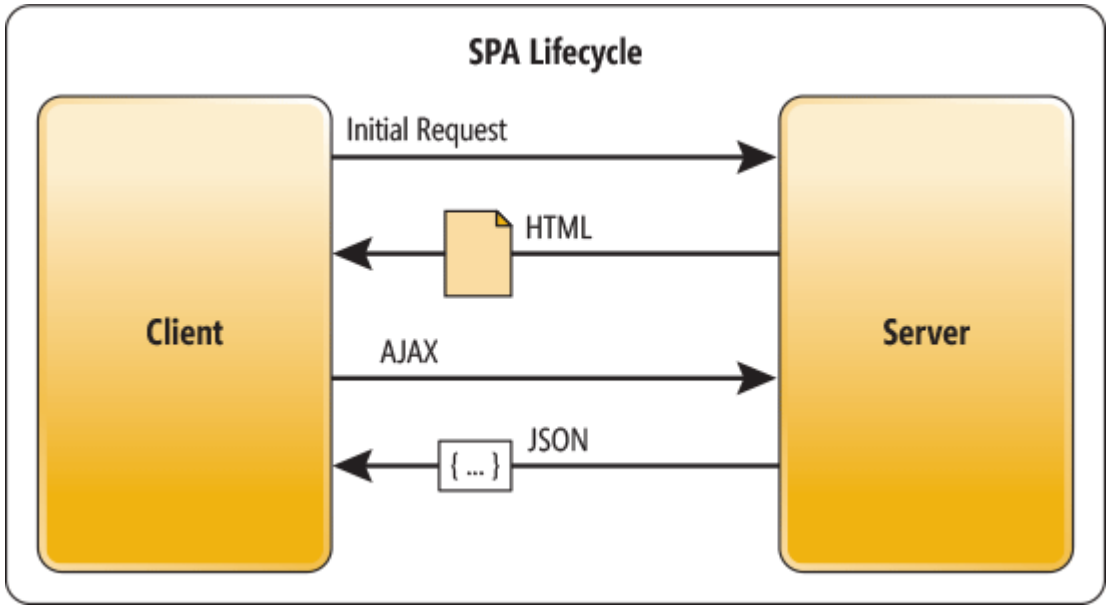
Traditional / MPA

- Each time app calls the server, server renders new HTML page
  - Triggers a page refresh in the browser



SPA

- In SPA, after the first page loads, all interaction with the server happens through AJAX
  - Return data usually in JSON



https://msdn.microsoft.com/en-us/magazine/dn463786.aspx

SPA advantages / disadvantages

SPA advantages

- Decoupled: you could replace the entire back end that runs the service, and as long as you don’t change the API, you won’t break the client. The reverse is also true
- Reusability
  - Easier to make a mobile application because the developer can reuse the same backend code
- Caching capabilities
  - (HTML+CSS+Scripts) are only loaded once
- Code separation
  - MPA: Frontend and backend development are tightly coupled

SPA disadvantages

- Search engine optimization
  - Search engines can do JS
- Browser history
  - Nowadays solved
- More complex dev tools
  - New JS tools evolving fast
- Can be slow to download because heavy client frameworks are required to be loaded to the client.

URL: b.socrative.com/student/

Room: HSR

Web Application – Go for the backend

- **Setup Controller (Intro)**
  - FS18: Java, FS19: golang
  - server.go in {home}/go/src/…
  - go get (dependencies)
  - go run server.go
- **Exercises: setup GoLand, run on your machine**
  - Default GOPATH
    - $HOME/go
    - %USERPROFILE%\go
- **Build executable file**
  - go build

```
package main

import (
        "github.com/julienschmidt/httprouter"
        "html/template"
        "log"
        "net/http"
)

type Model struct {
        Name string
}

func helloWorld(w http.ResponseWriter, r *http.Request, ps httprouter.Params) {
        template, _ := template.ParseFiles("index.gtpl")
        model := Model{Name: ps.ByName("name")}
        template.Execute(w, &model)
}

func main() {
        log.Print("Starting server...")
        router := httprouter.New()
        router.GET("/nospa/:name", helloWorld)
        log.Fatal(http.ListenAndServe(":8080", router))
}
```

Web Application - Router

- **Router vs. Dispatcher**
  - HTTP request comes in for a resource - Router provides necessary directions to reach the correct resource
  - Router parsing dynamic arguments in URL
  - Dispatching uses information from Routing step to create response
- **Built in vs. httprouter**
  - Static (exact) routes or ease of use
  - Comparison
  - mux: Package gorilla/mux implements a request router and dispatcher…

```
package main

import (
        "github.com/julienschmidt/httprouter"
        "html/template"
        "log"
        "net/http"
)

type Model struct {
        Name string
}

func helloWorld(w http.ResponseWriter, r *http.Request, ps httprouter.Params) {
        template, _ := template.ParseFiles("index.gtpl")
        model := Model{Name: ps.ByName("name")}
        template.Execute(w, &model)
}

func main() {
        log.Print("Starting server...")
        router := httprouter.New()
        router.GET("/nospa/:name", helloWorld)
        log.Fatal(http.ListenAndServe(":8080", router))
}
```

Web Application

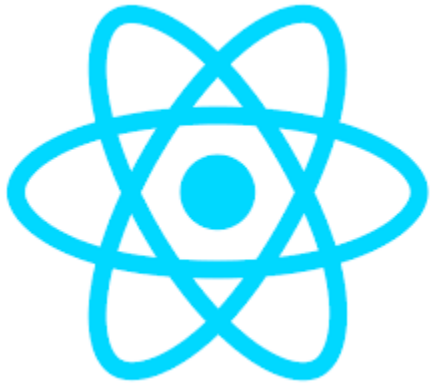
- **Server renders the HTML**
  - Fills in the variables
  - Sends HTML to browser
- **Template builtin**
  - Alternatives benchmark/list
  - Add go template templates/index.gtpl
  - css, javascript, HTML, chartset, icon,
- **Open http://localhost:8080/nospa/world**

```
<!DOCTYPE html>
<head>
   <meta charset="UTF-8">
   <title>Hello!</title>
</head>
<body>
<h2>Hello {{.Name}}!</h2>
</body>
</html>
```

Introduction – SPA Characteristics I

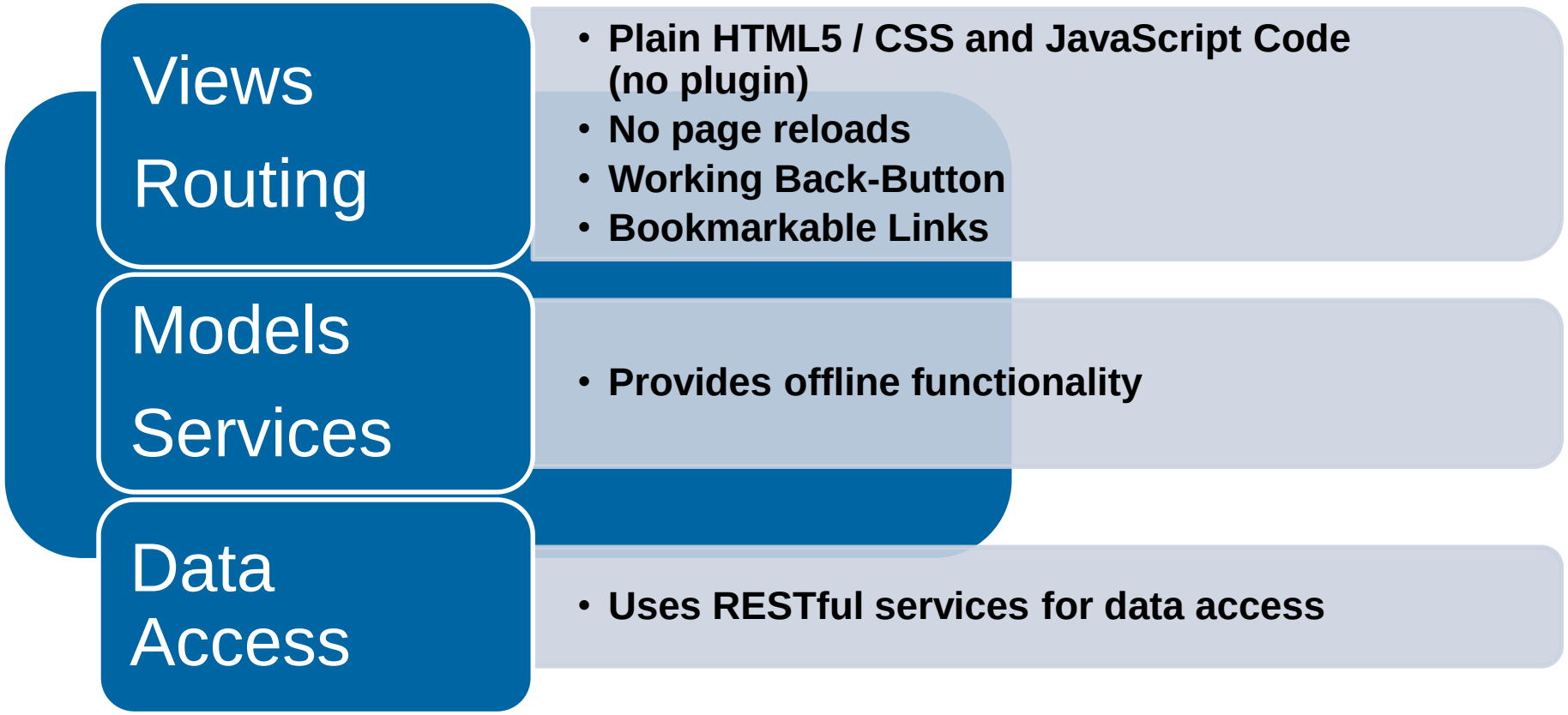
- **Plain HTML5 / CSS and JavaScript Code (no plugin)**
- **No page reloads**
- **Uses RESTful services for data access**
  - Ajax, websockets
- **Ideally**
  - Working Back-Button
  - Scroll position, cancel, unsaved changes
  - Bookmarkable Links
  - Provides offline functionality

- **AngularJS**
- **Ember.js**
- **ExtJS**
- **Knockout.js**
- **Meteor.js**
- **Aurelia**
- **Vue.js**
- **React**



Source: https://en.wikipedia.org/wiki/Single-page_application

Introduction – SPA Characteristics II



Single Page Application (SPA)

- **Simple SPA example**
  - golang / installation
  - IDE: goland / webstorm (free for students)
  - Visual Studio, Eclipse, Sublime, …
- **Server now returns JSON**
  - 2 new routes
  - 2 new functions
- **Plain Javascript**
  - Loaded in index.html
  - No framework

```
func index(w http.ResponseWriter, r *http.Request, _ httprouter.Params) {
        http.ServeFile(w, r, "index.html")
}

func spa(w http.ResponseWriter, _ *http.Request, ps httprouter.Params) {
        json.NewEncoder(w).Encode(Model{Name: ps.ByName("name")})
}
```

```
<!DOCTYPE html>
<head>
  <meta charset="UTF-8">
  <title>Hello!</title>

  <script type="text/javascript">
    fetch('/spa/'+window.location.hash.substring(1))
      .then(resp => resp.json())
      .then(data => {
        let element = document.getElementById('hello-title');
        element.innerHTML = data.Name;
      });
  </script>
</head>
<body>
  <h2 id="hello-title">empty</h2>
</body>
</html>
```

URL: b.socrative.com/student/

Room: HSR

State comparison

- **State Non-SPA**
  - On the server, in server.go

```
var counter = 0
…
func nospa(...)  {
   counter++
}
```

  - Session ID, cookies to store intermediate state,
  - Mapping session ID / data
  - No need for data binding on the client

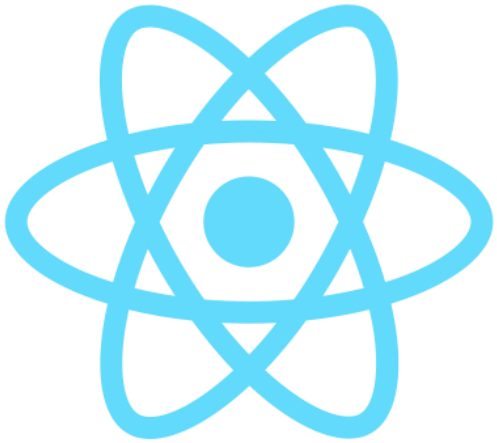
- **State SPA**
  - On the client, in index.html

```
<script type="text/javascript">
      counter = 0;
      …
      function send() {…
            counter++;
      }
```

  - No intermediate server side state
  - No mapping
  - Ease of use: data binding
  - More ease of use: components, client side routing, lifecycle management?

Vue.js Introduction

- Vue (pronounced /vjuː/, like view) is a progressive framework for building user interfaces.
- …On the other hand, Vue is also perfectly capable of powering sophisticated Single-Page Applications when used in combination with modern tooling and supporting libraries.
- 5 Best JavaScript Frameworks in 2017
  - **Angular, React, Vue**, Ember, Meteor
- Top 7 Most Popular JavaScript Frameworks in 2018
  - **Angular, React**, Ember, Aurelia, Meteor, Polymer, **Vue**
- **Vue only as an example in this lecture**



Vue.js Introduction

- **Simple Vue example**
  - Load Vue library
- **Vue built-in template syntax**
  - {{name}}
- **Reactive**
  - The data and the DOM are now linked, and everything is now reactive
  - vm.name = ‘hallo’

```
<!DOCTYPE html>
<head>
    <meta charset="UTF-8">
    <title>Hello!</title>
    <script src="https://cdn.jsdelivr.net/npm/vue"></script>
</head>
<body>
    <div id="app">
        <h2 class="hello-title">Hello {{name}}!</h2>
    </div>
    <script type="text/javascript">
        var vm = new Vue({
            el: '#app',
            data: {
                name: 'Hello Vue!'
            },
            created: function () {
                fetch('/spa/'+window.location.hash.substring(1))
                    .then(response => response.json())
                .then(data => {
                    let string = 'Markets:<br>';
                    for (let s of data) {
                        string += s.name + ', price=' + s.price_usd + '<br>';
                    }
                    this.name = string;
                });
            }
        });
    </script>
</body>
</html>
```

URL: b.socrative.com/student/

Room: HSR

Questions?

Prof. Dr. Thomas Bocek
thomas.bocek@hsr.ch