Kai Tödter

Cool Web Apps
with Spring Boot, Angular & TypeScript

Who am I?

- Principal Key Expert
  at Siemens Building Technologies
- Web Technology Fan
- Open Source Lover
- E-mail: kai@toedter.com
- Twitter: twitter.com/kaitoedter
- Blog: toedter.com/blog

Show Hands!

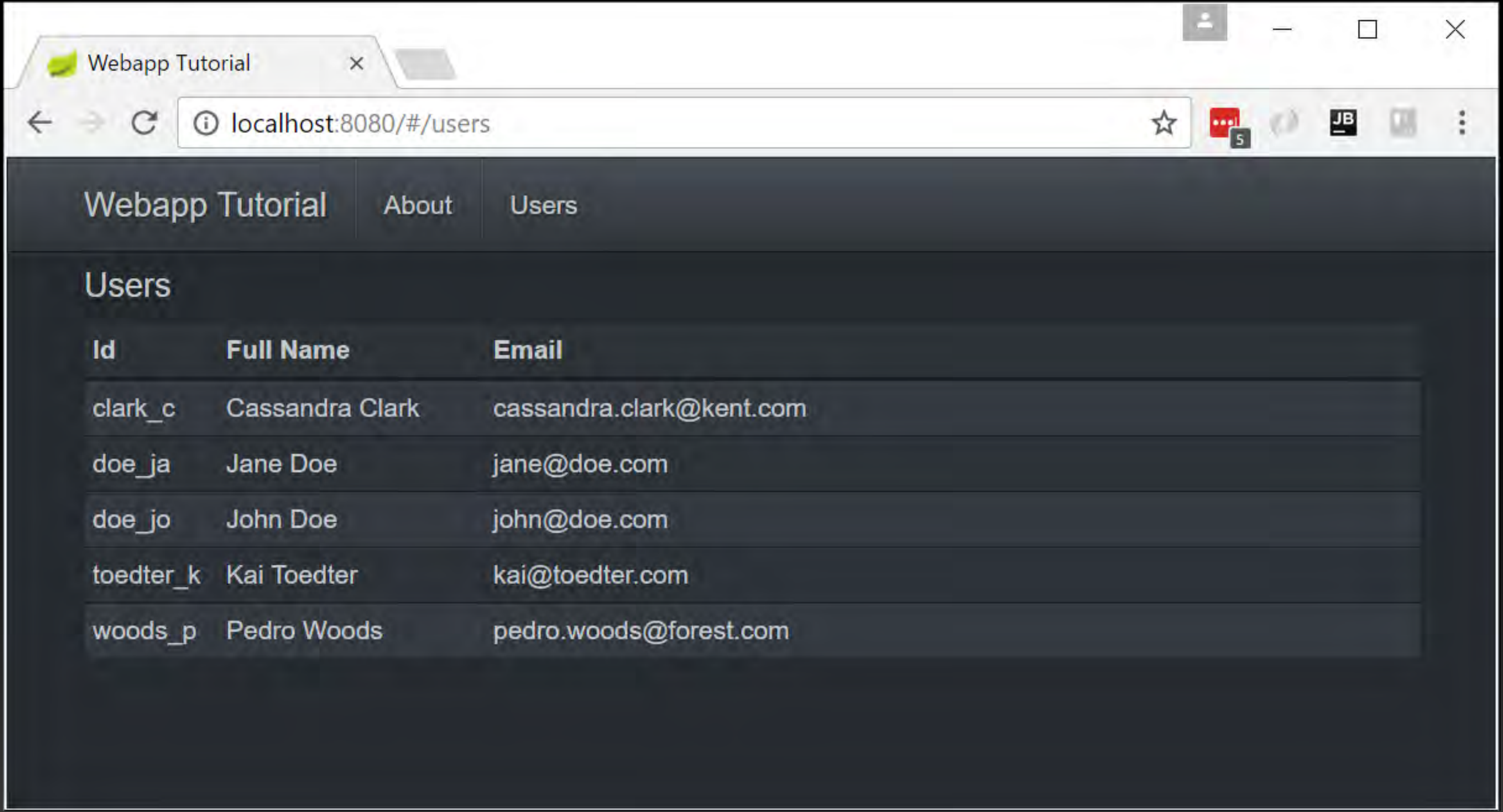
Spring Boot?
Hypermedia?
TypeScript?
Angular?
Bootstrap?

After this tutorial I'll have an idea how to put it all together!

Outline

- Spring Boot
- REST & Hypermedia
- Spring Data Rest
- TypeScript
- Angular
- Bootstrap
- Putting it all together

What we will create…



Webapp Tutorial ×

localhost:8080/#/users

Webapp Tutorial | About | Users

Users

Id	Full Name	Email
clark_c	Cassandra Clark	cassandra.clark@kent.com
doe_ja	Jane Doe	jane@doe.com
doe_jo	John Doe	john@doe.com
toedter_k	Kai Toedter	kai@toedter.com
woods_p	Pedro Woods	pedro.woods@forest.com

Requirements: Labs on local Machine

- Your favorite text editor or IDE
  - Eclipse, IntelliJ IDEA, Sublime, …
- Java 7 JDK or later installed
- Optional: Node.js installed

Requirements: Labs in Virtual Machine

- VirtualBox installed
  - See https://www.virtualbox.org/wiki/Downloads
- Get the latest Tutorial VM
  - Or install Vagrant and run “vagrant up” in the vagrant directory
- Start VM in VirtualBox
- Login in as “vagrant” with password “vagrant”
- Type “startx” to start the desktop

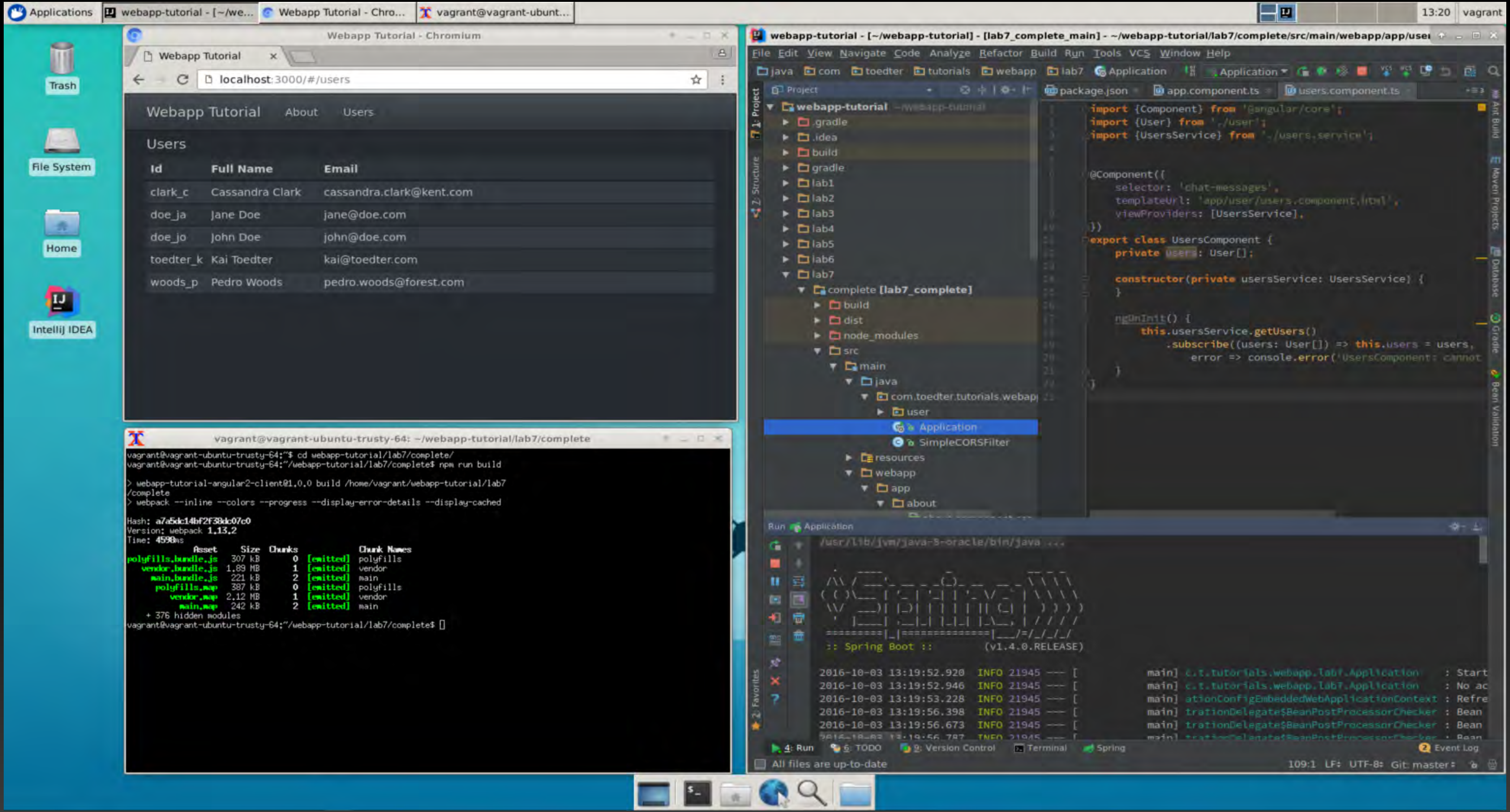
Lab 0: Build all tutorial labs locally

- Install the tutorial sources
  - Clone https://github.com/toedter/webapp-tutorial
  - Or copy from USB stick
- cd into webapp-tutorial
  - gradlew prepareJS build --console plain
  - Linux & Mac: ./gradlew instead of gradlew
- If the build is successful, you are ready to go!

Lab 0: Build all tutorial labs in the VM

- When using a VM created by Kai
  - Everything is setup already
- When having created the VM yourself using Vagrant
  - Open IntelliJ IDEA (you find it in /opt)
  - Configure Java 8 SDK (/usr/lib/jvm/...)
  - Import gradle project webapp-tutorial (in ~)
  - Install Lombok plugin in IntelliJ IDEA
  - In IntelliJ IDEA enable “Annotation Processing”
- In Chromium install extension “JSONView”

Prepared Tutorial VM



Webapp Tutorial   About   Users

Users

Id	Full Name	Email
clark_c	Cassandra Clark	cassandra.clark@kent.com
doe_ja	Jane Doe	jane@doe.com
doe_jo	John Doe	john@doe.com
toedter_k	Kai Toedter	kai@toedter.com
woods_p	Pedro Woods	pedro.woods@forest.com

```
vagrant@vagrant-ubuntu-trusty-64:~$ cd webapp-tutorial/lab7/complete/
vagrant@vagrant-ubuntu-trusty-64:~/webapp-tutorial/lab7/complete$ npm run build

> webapp-tutorial-angular2-client@1.0.0 build /home/vagrant/webapp-tutorial/lab7
/complete
> webpack --inline --colors --progress --display-error-details --display-cached

Hash: a7a5dc14bf2f38dc07c0
Version: webpack 1.13.2
Time: 4598ms
             Asset     Size  Chunks             Chunk Names
polyfills.bundle.js   307 kB       0  [emitted]  polyfills
   vendor.bundle.js  1.89 MB       1  [emitted]  vendor
     main.bundle.js   221 kB       2  [emitted]  main
      polyfills.map   387 kB       0  [emitted]  polyfills
         vendor.map  2.12 MB       1  [emitted]  vendor
           main.map   242 kB       2  [emitted]  main
    + 376 hidden modules
vagrant@vagrant-ubuntu-trusty-64:~/webapp-tutorial/lab7/complete$ 
```

```
import {Component} from '@angular/core';
import {User} from './user';
import {UsersService} from './users.service';


@Component({
    selector: 'chat-messages',
    templateUrl: 'app/user/users.component.html',
    viewProviders: [UsersService],
})
export class UsersComponent {
    private users: User[];

    constructor(private usersService: UsersService) {
    }

    ngOnInit() {
        this.usersService.getUsers()
            .subscribe((users: User[]) => this.users = users,
                error => console.error('UsersComponent: cannot
    }
}
```

```
/usr/lib/jvm/java-8-oracle/bin/java ...

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v1.4.0.RELEASE)

2016-10-03 13:19:52.920  INFO 21945 --- [           main] c.t.tutorials.webapp.lab7.Application    : Start
2016-10-03 13:19:52.946  INFO 21945 --- [           main] c.t.tutorials.webapp.lab7.Application    : No ac
2016-10-03 13:19:53.228  INFO 21945 --- [           main] ationConfigEmbeddedWebApplicationContext : Refre
2016-10-03 13:19:56.398  INFO 21945 --- [           main] trationDelegate$BeanPostProcessorChecker : Bean
2016-10-03 13:19:56.673  INFO 21945 --- [           main] trationDelegate$BeanPostProcessorChecker : Bean
```

Spring Boot

Why Spring Boot?

- Fast way to build web applications
- Inspects your classpath and beans you have configured
- You can focus more on business features and less on infrastructure
- Easily deployable as microservice

What does Spring Boot NOT?

- Generate code
- Change your configuration

HelloController

```
@RestController
public class HelloController {

    @RequestMapping("/")
    public String index() {
        return "Greetings from Spring Boot!";
    }
}
```

Application

```
@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

Controller Test with MockMVC

```
@RunWith(SpringRunner.class)
@WebMvcTest(HelloController.class)
public class HelloControllerTest {

    @Autowired
    private MockMvc mockMVC;

    @Test
    public void shouldGetGreeting() throws Exception {
        mockMVC.perform(MockMvcRequestBuilders.get("/")
                .accept(MediaType.APPLICATION_JSON))
                .andExpect(status().isOk())
                .andExpect(content().string(equalTo(
                    HelloController.LAB1_GREETINGS_FROM_SPRING_BOOT)));
    }
}
```

Integration Test with Random Port

```
@RunWith(SpringRunner.class)
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM_PORT)
public class HelloControllerIntegrationTest {

  @Autowired
  private TestRestTemplate restTemplate;

  @Test
  public void shouldGetGreeting() throws Exception {
    ResponseEntity<String> response = restTemplate.getForEntity("/", String.class);
    assertThat(response.getBody(),
              equalTo(HelloController.LAB1_GREETINGS_FROM_SPRING_BOOT));
  }
}
```

Lab 1: Task 1

- Open terminal in lab1/complete
- Invoke ..\..\gradlew bootrun
  - On Linux & Mac: sudo ../../gradlew bootrun
- Open browser with localhost:8080

Lab 1: Task 2

- Open lab1/initial in your IDE
- Create a Spring Boot based web app
- Run it and open your browser with localhost:8080
- Optional: Write some tests!
  - Get some ideas from …/complete

REST + Hypermedia Basics

Outline

- REST Basics
- HATEOAS
- Hypermedia with HAL
- Spring Data Rest

What is REST?

- Stands for Representational State Transfer
- Is a Software Architecture Style
- was introduced and defined in 2000 by Roy T. Fielding in his doctoral dissertation

- REST != CRUD via HTTP

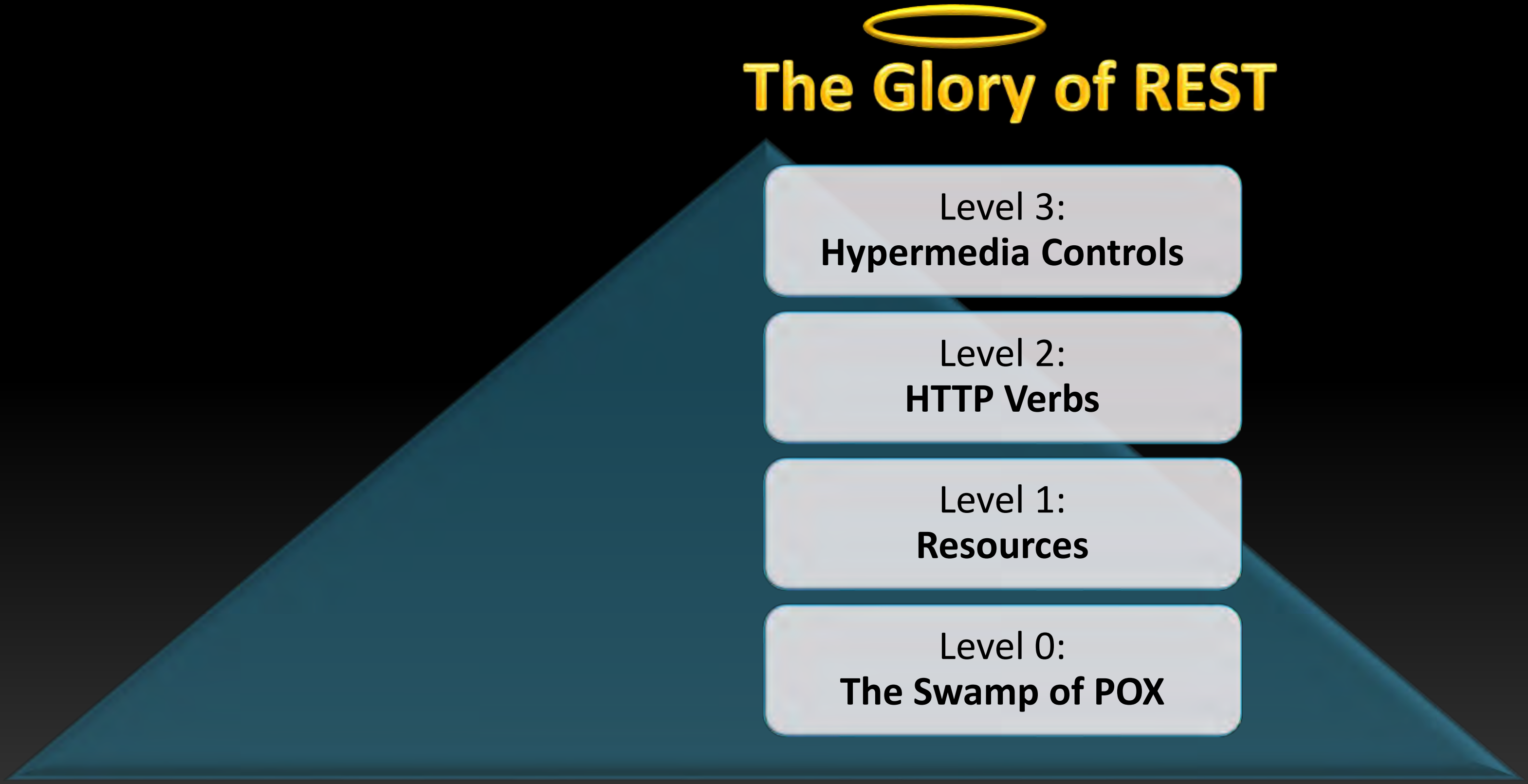
REST Architectural Constraints

- Client-Server
- Stateless
- Cacheable
- Layered system
- Code on demand (optional)
- Uniform interface (see next slide)

Uniform Interface

- Identification of resources
- Manipulation of resources through their representations
  - Create => HTTP POST
  - Read => HTTP GET
  - Update => HTTP PUT, HTTP PATCH
  - Delete => HTTP DELETE
- Self-descriptive messages
- Hypermedia as the engine of application state (HATEOAS)

Richardson Maturity Model



See http://martinfowler.com/articles/richardsonMaturityModel.html

Hypermedia APIs

for Services

are like

Web Pages with Links

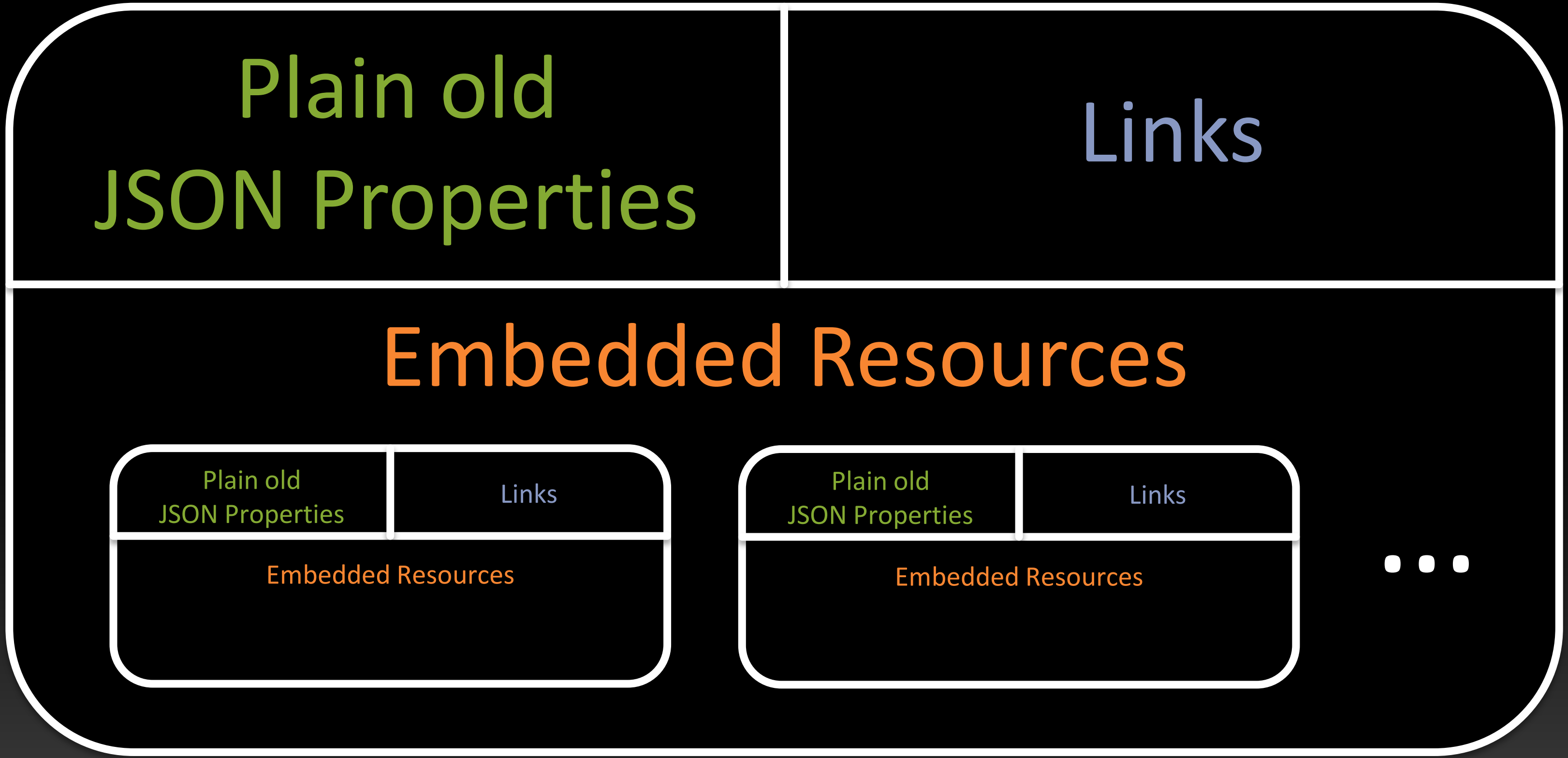
for Humans

HAL

HAL

- Is for Hypertext Application Language
- Was created by Mike Kelly
- Representations for both JSON and XML
- Very popular

HAL Structure


Plain old JSON Properties
Links
Embedded Resources
Plain old JSON Properties
Links
Embedded Resources
Plain old JSON Properties
Links
Embedded Resources
...

HAL Example

```
{
   "id":1,
   "text":"hello all!",
   "_links": {
      "self": {
         "href":"http://localhost:8080/chatty/api/messages/1"
       }
   },
   "_embedded": {
      "author": {
          "id":"toedter_k"
       }
    }
}
```

Spring

- Spring Boot
- Spring Data Rest
- Spring HATEOAS

Spring Data Rest: Domain

```
@Data
@Entity
@NoArgsConstructor
public class User {
    @Id
    private String id;
    private String fullName;
    private String email;
}
```

Spring Data REST: Repository

```
@RepositoryRestResource(
        collectionResourceRel = "users",
        path = "users")

interface UserRepository extends
         PagingAndSortingRepository<User, String> {
}
```

Spring Data REST: Repository (2)

```
@RepositoryRestResource( exported = false )

interface UserRepository extends
        PagingAndSortingRepository<User, String> {
}
```

Spring Data Rest: JSON Result

```
{
  _links: {
    self: {
      href: "http://localhost:8080/chatty/api/users{?page,size,sort}",
      templated: true
    }
  },
  _embedded: {
     users: [ {
       fullName: "Jane Doe",
       email: "jane@doe.com",
       _links:  {
         self: {
            href: "http://localhost:8080/chatty/api/users/doe_ja",
            templated: true
         },
…
```

Robust Clients

- Start from main API
- Find link relations through defined contracts
- Follow Links
  - For navigation
  - For possible “actions”

=> Clients are robust regarding changes in link URIs

Controversial Discussion

- Are we there yet?
- RESTistential Crises
  - http://www.infoq.com/news/2014/03/rest-at-odds-with-web-apis
- DHH, Getting hyper about hypermedia apis
  - https://signalvnoise.com/posts/3373-getting-hyper-about-hypermedia-apis

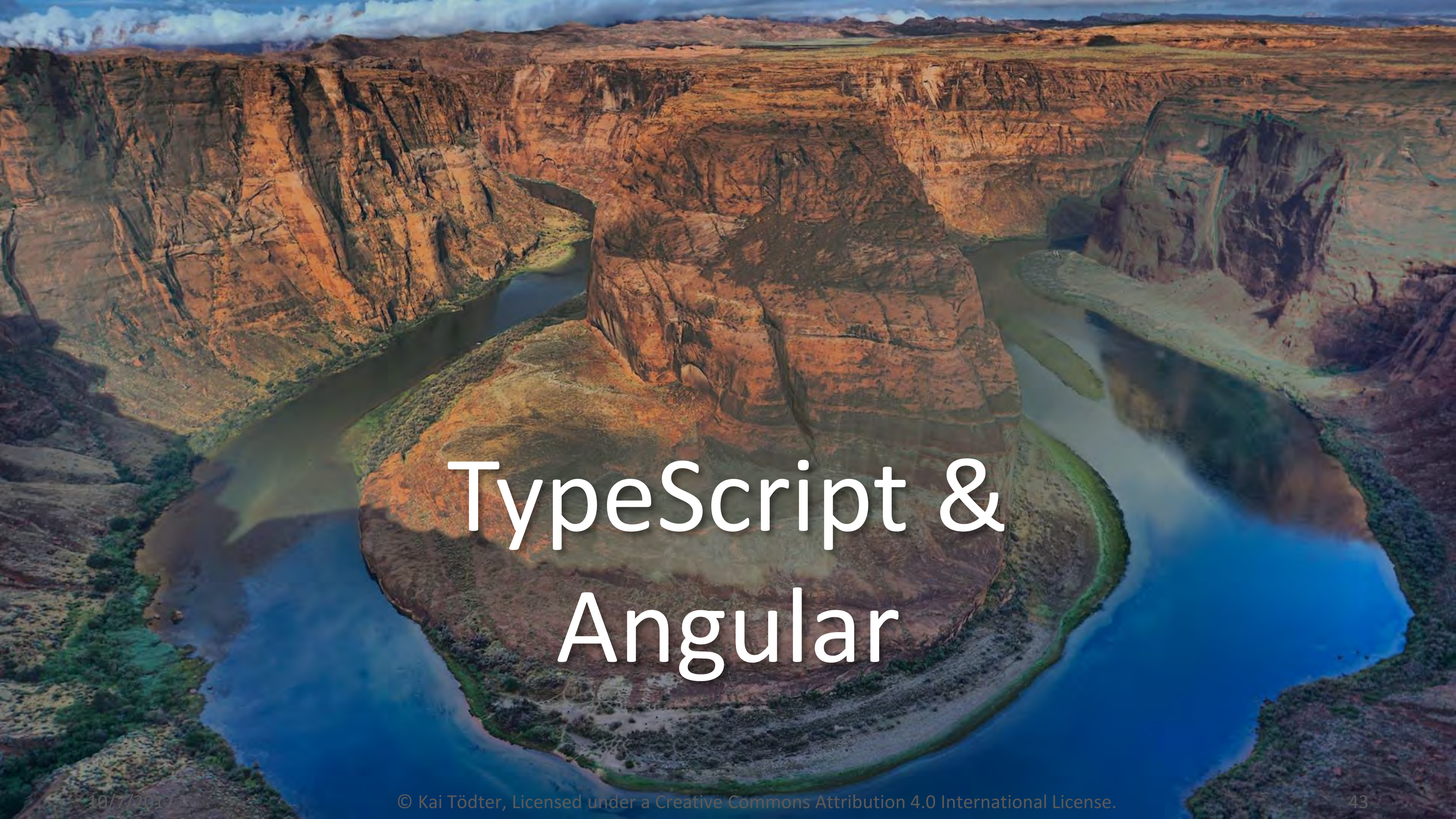
Live Demo + Tests

Lab 2: Task 1

- Open terminal in lab2/complete
- Invoke ..\..\gradlew bootrun
- Open browser with localhost:8080

Lab 2: Task 2

- Open lab2/initial in your IDE
- Add a user repository
- Fill the repository with test data
- Run the application and open your browser with localhost:8080
- Optional: Write some tests!
  - Get some ideas from ../complete



TypeScript & Angular

Outline

- TypeScript Introduction
- Angular Introduction
- TypeScript + Angular
- Demos & Live Coding

JavaScript?

Many Java/OO developers don’t like JavaScript regarding writing larger applications. Some reasons are:

- No static typing
  - No reliable code completion (only best guess)
  - Hard to refactor
- Not object-oriented, especially
- No structuring mechanisms like Interfaces, Classes*, Modules*

* Before ECMAScript 2015

Who fixes that?

- Dart
  - Great language by Google: dartlang.org
  - Team has to learn new language
  - Either runs on Dart VM or compiles to JavaScript
- CoffeeScript
  - Ruby-like, concise syntax
  - Compiles to JavaScript
  - coffescript.org
- BabelJS
  - JavaScript compiler
  - babeljs.io
- Traceur
  - JavaScript compiler
  - github.com/google/traceur-compiler

TypeScript: Summary

- Typed Superset of JavaScript
  - Almost all valid JavaScript is valid TypeScript*
- Compiles to JavaScript
- Provides optional static type checking at compile time
  - For most existing JavaScript libraries there are type definitions available
- Provides Interfaces, Classes, Modules, Enums, Generics, Decorators and more
- Open Source: Apache 2.0 License
- Created by Microsoft

How to get started?

- www.typescriptlang.org
- Install Node.js (nodejs.org)
- Invoke “npm install –g typescript”
- Compile a TypeScript file:
  “tsc myTypeScript.ts”
  - Results in “myTypeScript.js”

www.typescriptlang.org

TypeScript

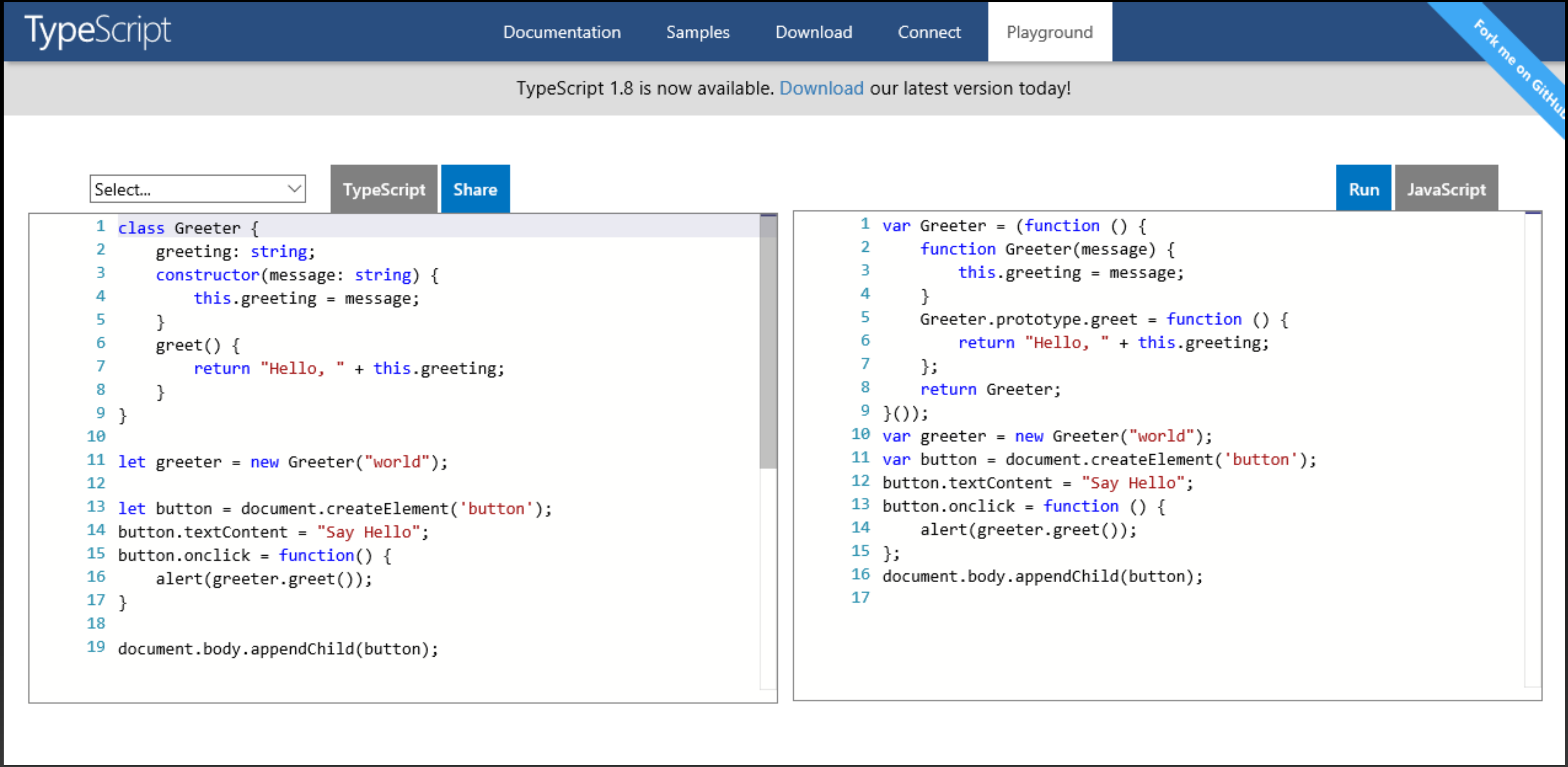
JavaScript that scales.

TypeScript is a typed superset of JavaScript that compiles to plain JavaScript.

Any browser. Any host. Any OS. Open source.

Download
Documentation
Starts and ends with JavaScript
Strong tools for large apps
State of the art JavaScript

Play!

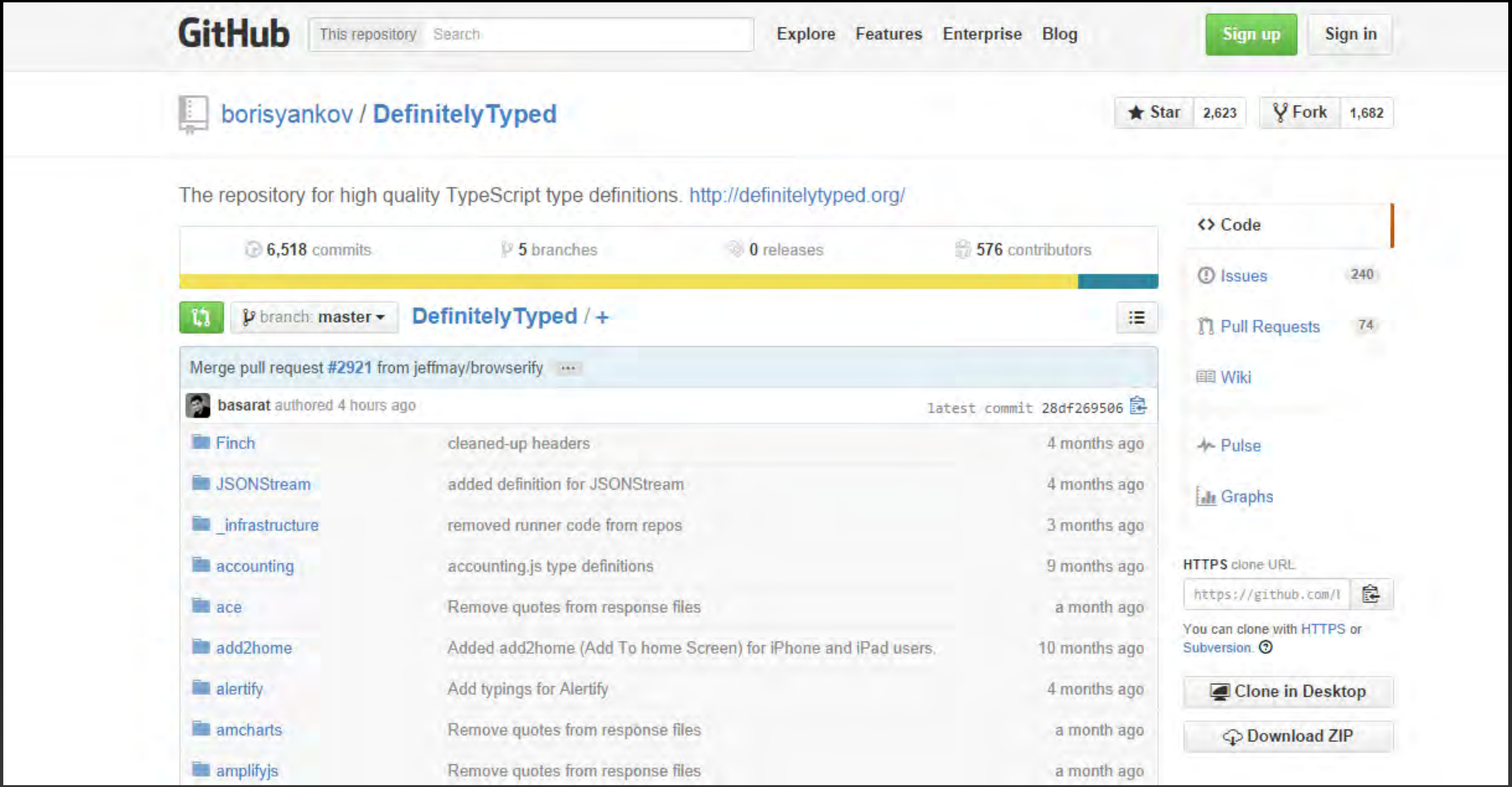


TypeScript 1.8 is now available. Download our latest version today!

```typescript
class Greeter {
    greeting: string;
    constructor(message: string) {
        this.greeting = message;
    }
    greet() {
        return "Hello, " + this.greeting;
    }
}

let greeter = new Greeter("world");

let button = document.createElement('button');
button.textContent = "Say Hello";
button.onclick = function() {
    alert(greeter.greet());
}

document.body.appendChild(button);
```

```javascript
var Greeter = (function () {
    function Greeter(message) {
        this.greeting = message;
    }
    Greeter.prototype.greet = function () {
        return "Hello, " + this.greeting;
    };
    return Greeter;
}());
var greeter = new Greeter("world");
var button = document.createElement('button');
button.textContent = "Say Hello";
button.onclick = function () {
    alert(greeter.greet());
};
document.body.appendChild(button);
```

Definitely Typed



GitHub | This repository | Search | Explore | Features | Enterprise | Blog | Sign up | Sign in

borisyankov / DefinitelyTyped

Star 2,623 | Fork 1,682

The repository for high quality TypeScript type definitions. http://definitelytyped.org/

6,518 commits | 5 branches | 0 releases | 576 contributors

branch: master | DefinitelyTyped / +

Merge pull request #2921 from jeffmay/browserify

basarat authored 4 hours ago — latest commit 28df269506

Finch	cleaned-up headers	4 months ago
JSONStream	added definition for JSONStream	4 months ago
_infrastructure	removed runner code from repos	3 months ago
accounting	accounting.js type definitions	9 months ago
ace	Remove quotes from response files	a month ago
add2home	Added add2home (Add To home Screen) for iPhone and iPad users.	10 months ago
alertify	Add typings for Alertify	4 months ago
amcharts	Remove quotes from response files	a month ago
amplifyjs	Remove quotes from response files	a month ago

Namespaces and Interfaces

```
namespace tutorial.webapp {

    export interface User {
        getId(): string;
        getEmail(): string;
        getFullName(): string;
    }
}
```

Classes

```
namespace tutorial.webapp {
    export class SimpleUser implements User {
        constructor(private id: string,
                    private email: string,
                    private fullName: string) { }

        getId(): string {
            return this.id;
        }
...
```

Live Demo

JavaScript Dev Tools

JavaScript Dev Tools

- In JavaScript land, mostly JavaScript based tools are used for build, dependency management, test, etc.
  - npm for
    - dependency management (including @types)
    - Running build/test/server scripts
  - Jasmine for implementing tests
  - Karma for running tests

npm

- Package manager for JavaScript
- Resolves dependencies
- Runs scripts
- Is THE JavaScript dev tool
- Input file is package.json

package.json Example (1)

```
{
  "name": "tutorial-web-client",
  "title": "tutorial web client",
  "version": "1.0.0",
  "description": "tutorial web client - a tutorial lab",
  "scripts": {
    "build": "./node_modules/.bin/tsc",
    "test": "./node_modules/.bin/karma start"
  },
  "author": {
    "name": "Kai Toedter",
    "url": "http://toedter.com"
  },
```

package.json Example (2)

```
  "license": "MIT",
  "dependencies": {
  },
  "devDependencies": {
    "jasmine-core": "2.8.0",
    "karma": "1.7.1",
    "karma-jasmine": "1.1.0",
    "karma-phantomjs-launcher": "1.0.4",
    "typescript": "2.5.2"
}
```

Jasmine Example

```
describe('User', () => {
    it('should create user and get attributes', () => {
        var user:User =
            new SimpleUser("user", "user@test.com", "User 1");
        expect(user).toBeDefined();
        expect(user.getId()).toBe('user1');
        expect(user.getEmail()).toBe('user1@test.com');
        expect(user.getFullName()).toBe('User 1');
    });
});
```

Since TypeScript 2.0: @types

- All typings are available as npm modules
- Install a typing with
  - npm install @types/<npm module>
  - E.g. npm install @types/jasmine

@types in tsconfig.json

```
{
  "compilerOptions": {
    "module": "commonjs",
    "target": "es5",
    "outDir": "build/dist",
    "rootDir": ".",
    "sourceMap": true,
    "emitDecoratorMetadata": true,
    "experimentalDecorators": true,
    "moduleResolution": "node",
    "typeRoots": [
      "node_modules/@types"
    ]
  }
}
```

Webpack

- https://github.com/webpack/webpack

webpack
MODULE BUNDLER

Webpack

- Webpack is a bundler for modules
  - bundles JavaScript files for usage in a browser
  - can transform, bundle, or package any resource or asset
- Bundles both CommonJS and AMD modules
- Can create a single bundle or multiple chunks
- Dependencies are resolved during compilation
  - reduces the runtime size
- Loaders can preprocess files while compiling
  - e.g. TypeScript to JavaScript
- Highly modular plugin system

Webpack Config (1)

```
var webpackConfig = {
    entry: {
        'main': './src/main/webapp/main.browser.ts'
    },

    output: {
        publicPath: '',
        path: path.resolve(__dirname, './build/dist'),
    },

    …
```

Webpack Config (2)

```
…
plugins: [
    new webpack.ContextReplacementPlugin(
        /angular(\\|\/)core(\\|\/)src(\\|\/)linker/,
        path.resolve(__dirname, './src'), { }
    )
],
module: {
    loaders: [
        { test: /\.ts$/, loaders: ['awesome-typescript-loader', 'angular2-template-loader' ]},
        { test: /\.css$/, loaders: ['to-string-loader', 'css-loader'] },
        { test: /\.html$/, loader: 'raw-loader' }
    ]
}
```

Lab 3: Task 1

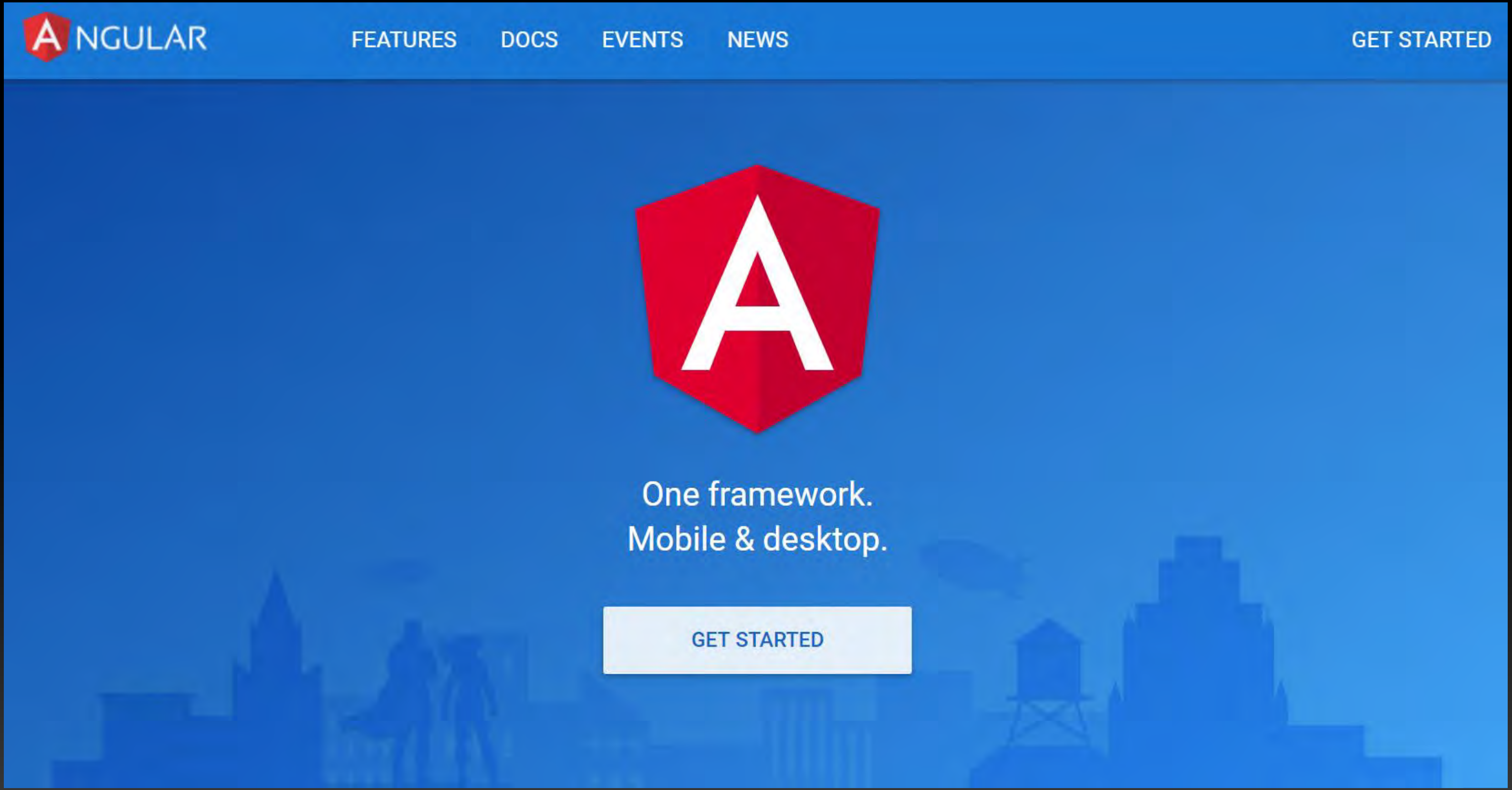
- Open terminal in lab3/complete
- Invoke npm run build
  - Check that all TypeScript files were transpiled to JavaScript in the dist directory
- Invoke npm test
  - Check that the Karma run was successful
  - And all Jasmine tests are green

Lab 3: Task 2

- Complete initial implementation of
  - User.ts, SimpleUser.ts, TestData.ts
  - UserSpec.ts
- Open terminal in lab3/initial
  - Invoke npm run build
    - Check everything builds
  - Invoke npm test
    - Check that all tests are green

ANGULAR

angular.io


ANGULAR
FEATURES
DOCS
EVENTS
NEWS
GET STARTED
One framework.
Mobile & desktop.
GET STARTED

Angular

- Angular is a framework for building client applications in HTML
- TypeScript, JavaScript, Dart
- Modules, Components, Templates, Services
- Much more…

Modules

- Every Angular app has at least one module, the *root module*
- Conventionally named AppModule
- A module is a class with an @NgModule decorator

@NgModule

- declarations
  - view classes (components, directives, and pipes) of this module
- exports
  - subset of declarations usable by other modules
- imports
  - exported classes of other modules needed by component templates *this* module's templates
- providers
  - creators of services (globally accessible in all parts of the app)
- bootstrap
  - main application view (root component)
  - Only the root module should set this bootstrap property

Example Module

```
import {NgModule} from '@angular/core';
import {AppComponent} from './app.component';
import {BrowserModule} from '@angular/platform-browser';
import {LocationStrategy, HashLocationStrategy} from '@angular/common';

@NgModule({
  declarations: [AppComponent],
  imports       : [BrowserModule],
  providers     : [{provide: LocationStrategy, useClass: HashLocationStrategy}],
  bootstrap     : [AppComponent]
})
export class AppModule {}
```

Angular Libraries

- Angular ships as a collection of Node.js modules
- Can be found in node_modules/@angular
  - common
  - compiler
  - core
  - forms
  - http
  - platform-browser
  - platform-browser-dynamic
  - router

Launching the App

- … by bootstrapping its root module

```
import {platformBrowserDynamic} from '@angular/platform-browser-dynamic';
import {AppModule} from './app/app.module';

platformBrowserDynamic().bootstrapModule(AppModule)
  .catch(err => console.error(err));
```

Components

- A component controls a view
- In our tutorial, we have 3 components:
  - AppComponent
  - UsersComponent
  - AboutComponent

@Component

- Takes configuration to
  - create and present the component and its view.
- A few @Component configuration options:
  - selector
    - CSS selector for this component
    - Here: <chat-messages></chat-messages>
    - Angular renders the templateURl between those tags
  - templateUrl
    - address of this component's HTML template
  - providers
    - array of dependency injection providers for services that the component requires

Component Example

```
import {Component} from '@angular/core';
import {User} from './user';
import {UsersService} from './users.service';

@Component({
   selector: 'chat-messages',
   templateUrl: 'app/user/users.component.html',
   providers: [UsersService],
})
export class UsersComponent  implements OnInit {
   private users: User[];

   constructor(private usersService: UsersService) {
   }

   ngOnInit() {
      this.users = this.usersService.getUsers();
   }
}
```

Templates

- A component's view is defined by a template
- A template looks like regular HTML,
  but can use Angular specific things

User List Template

```
<div>
  <h4>Users</h4>
  <table>
    <tr *ngFor="let user of users">
      <td>{{user.id}}</td>
      <td>{{user.fullName}}</td>
      <td>{{user.email}}</td>
  </table>
</div>
```

One Way Databinding

- Component -> DOM
  - interpolation: {{user.id}}
  - Property binding:
    - <user-detail [user]="selectedUser"></user-detail>
- DOM -> Component
  - <li (click)="selectUser(user)"></li>

Two Way Databinding

- Component <-> DOM
  - `<input [(ngModel)]="user.name">`

Recommendation:

Prefer one way databinding, since it makes control flows easier to understand

Dependency Injection

- Used everywhere in Angular
- *injector* is the main mechanism
  - An injector maintains a *container* of service instances
  - service instances are created automatically by the injector
  - An injector can create a new service instance from a *provider*
- A *provider* is a recipe for creating a service
- Register *providers* with injectors
  - Either in modules
  - Or in components

Services

- Are POTOs (Plain Ole Typescript Objects)
- Should encapsulate functionality that can be uses by
  - Other services
  - Components

Example Service

```
import {Injectable} from '@angular/core';
import {User} from './user';

@Injectable()
export class UsersService {
    public getUsers(): User[] {
        return [
            new User('toedter_k', 'Kai Toedter', 'kai@toedter.com'),
            new User('doe_jo', 'John Doe', 'john@doe.com'),
            new User('doe_ja', 'Jane Doe', 'jane@doe.com')
        ];
    }
}
```

@Injectable

- Needed by services, that want to have other services injected
- Recommendation: Use it for every service, even if it would not be necessary. Why?
    - Future proofing:
      No need to remember @Injectable() when you add a dependency later
    - Consistency: All services follow the same rules, and you don't have to wonder why a decorator is missing

Routing

- Tell the router how to compose navigation URLs, set base in index.html, e.g. `<base href="/">`
- Import RouterModule and Routes in TypeScript
- Create a router configuration
- Use tag <router-outlet> to display routed components

Router Configuration

```
import { Routes } from '@angular/router';

import {UsersComponent} from './user/users.component';
import {AboutComponent} from './about/about.component';

export const routerConfig: Routes = [
    { path: '', redirectTo: 'users', pathMatch: 'full' },
    { path: 'users', component: UsersComponent },
    { path: 'about', component: AboutComponent }
];
```

Routing HTML

```html
<nav>
  <ul>
    <li><a [routerLink]="['about']">About</a></li>
    <li><a [routerLink]="['users']">Users</a></li>
  </ul>
</nav>
<router-outlet></router-outlet>
```

Live Demo

Lab 4: Task 1

- Open terminal in lab4/complete
- Invoke: npm start
  - builds everything using webpack
  - Starts a Web server at port 3000
- Open http://localhost:3000 in a web browser
- You should see “Hello, Angular2!”

Lab 4: Task 2

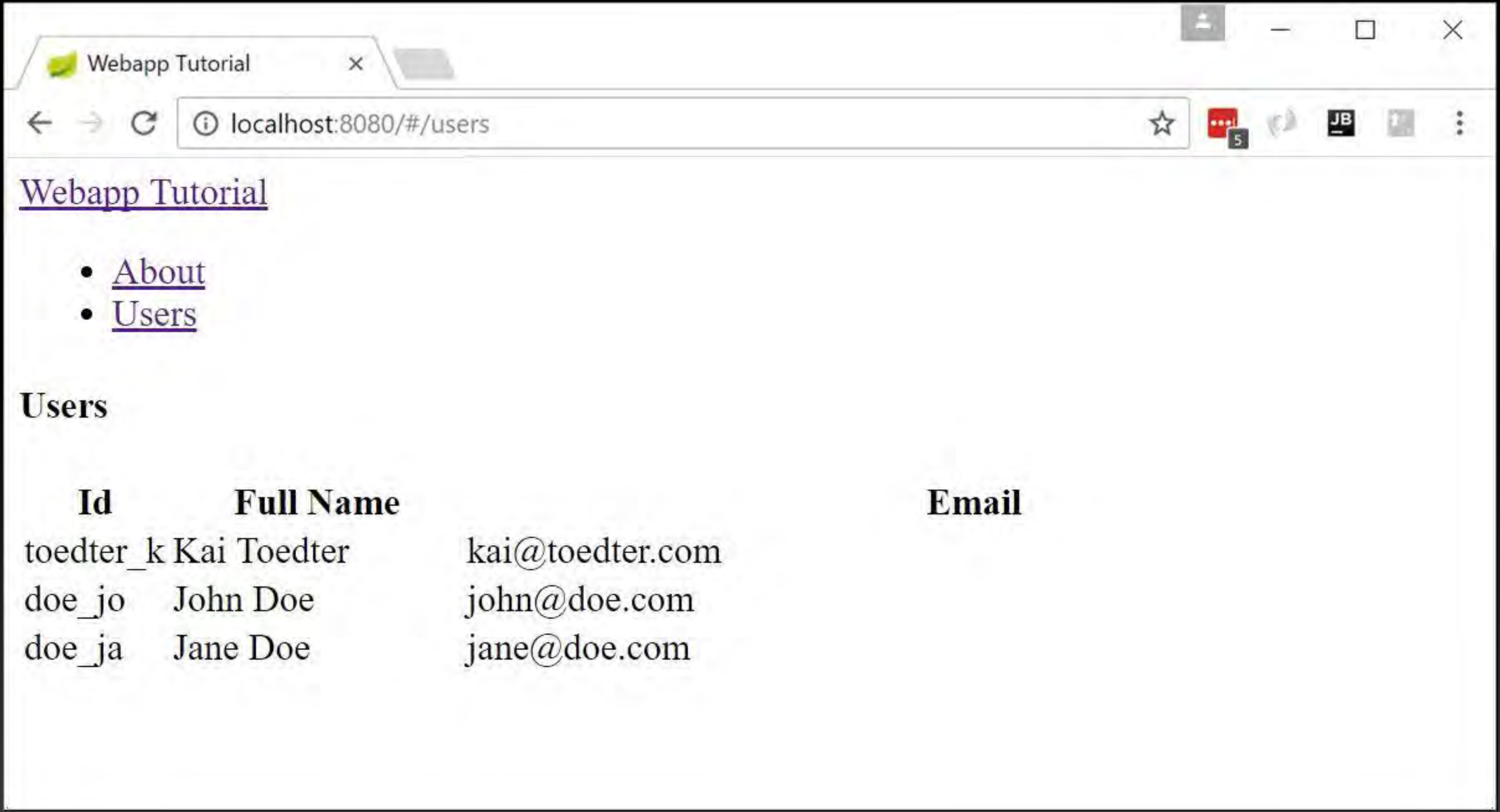
- Work in lab4/initial
- Implement a small Angular app that displays “hello <your name>”
  - Add decorators, constructor and method ngOnInit in app.component.ts
  - Add some Angular markup in app.component.html

Building the Angular App

Angular & TypeScript

- Angular is written itself in TypeScript
- Angular brings its own type definitions
- TypeScript is the recommended language to write Angular apps
  - But you could also use JavaScript or Dart

Angular App with no Styling



Webapp Tutorial

- About
- Users

Users

Id	Full Name	Email
toedter_k	Kai Toedter	kai@toedter.com
doe_jo	John Doe	john@doe.com
doe_ja	Jane Doe	jane@doe.com

Lab 5: Task 1

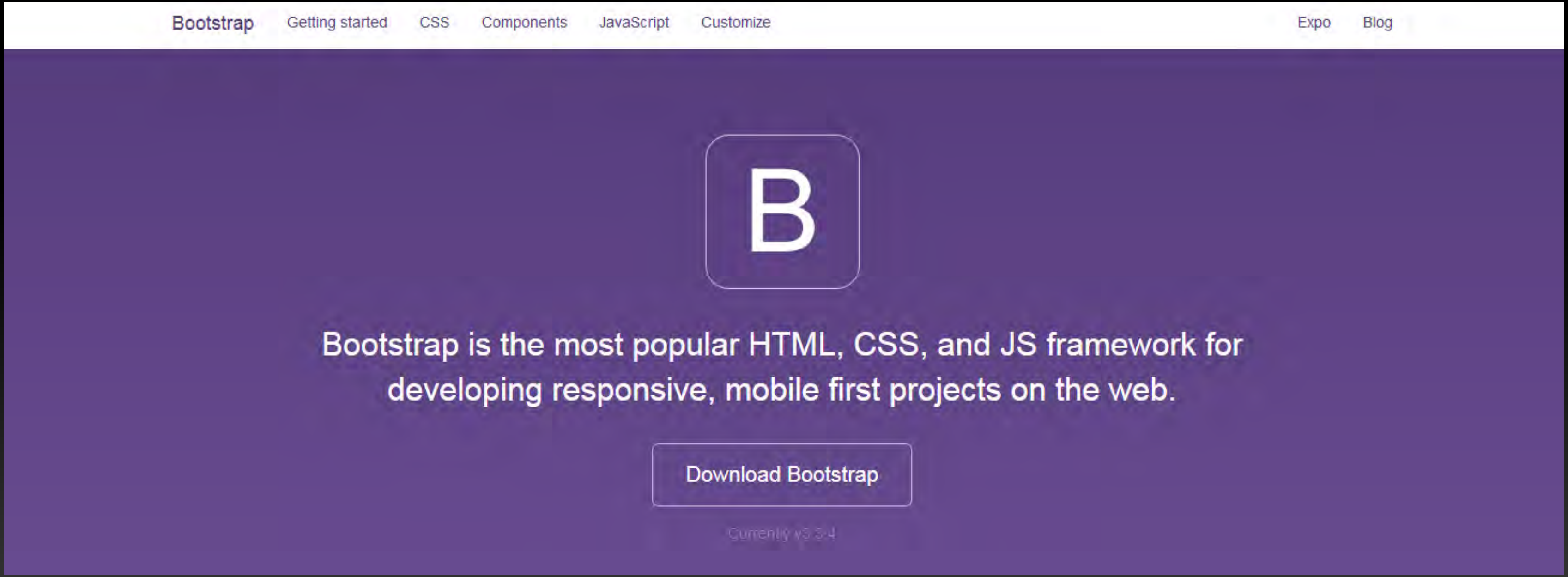
- Open terminal in lab5/complete
- Invoke: npm start
  - builds everything using webpack
  - Starts a Web server at port 3000
- Open http://localhost:3000 in a web browser
- You should see the web app with no styling
- Play around with it

Lab 5: Task 2

- Work in lab5/initial
- Implement a small Angular app that displays a list of users and an “About”
  - Add decorators, constructor and method ngOnInit in user.component.ts
  - Add ngular markup (*ngFor etc) in user.component.html

Bootstrap

- www.getbootstrap.com

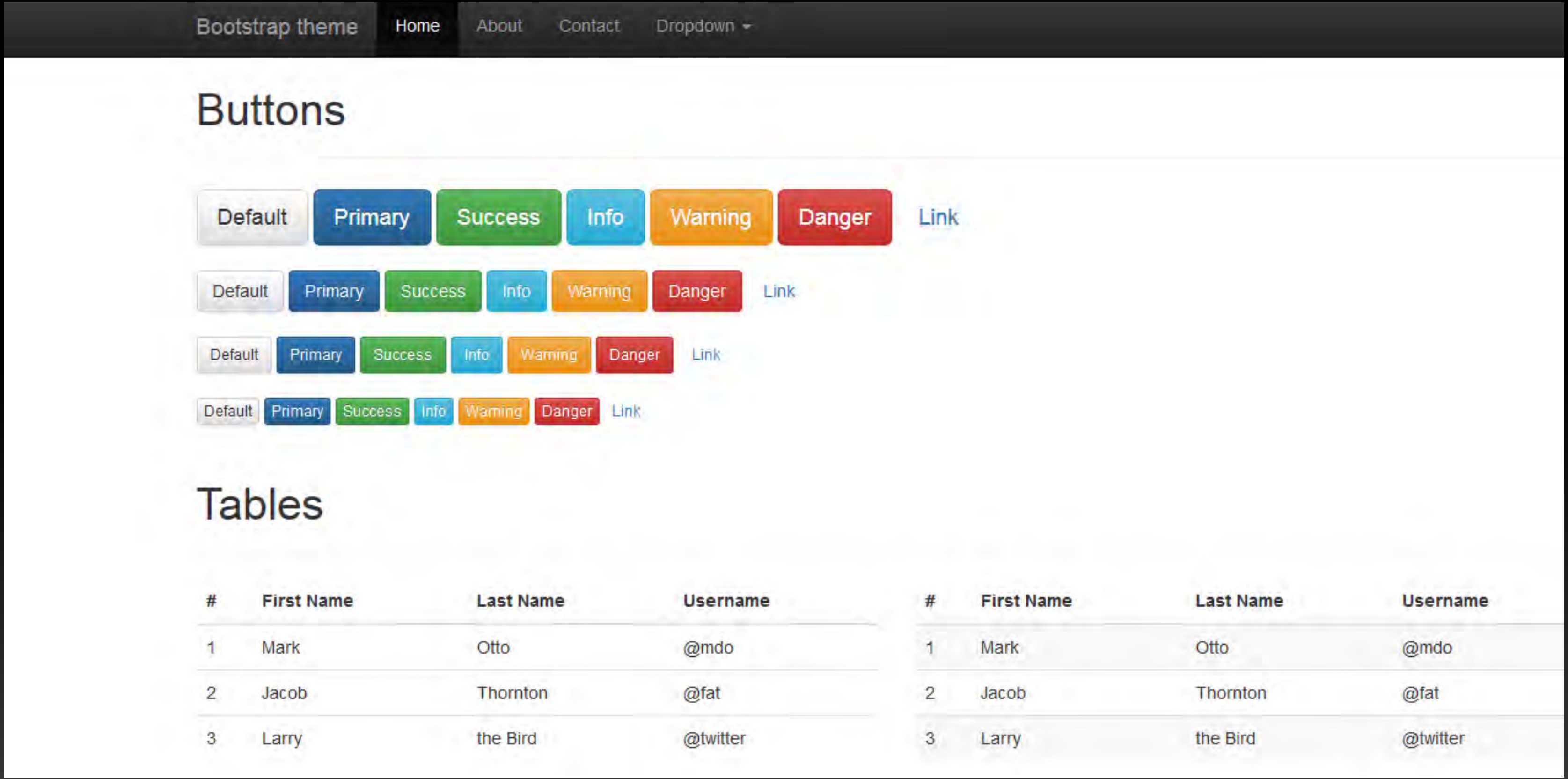


Bootstrap is the most popular HTML, CSS, and JS framework for developing responsive, mobile first projects on the web.

Bootstrap Summary

- By Twitter
- HTML, CSS3, JavaScript
- Templates
- Easy to use
- Mobile first

Bootstrap Live Demo


Bootstrap theme
Home
About
Contact
Dropdown

Buttons

Default Primary Success Info Warning Danger Link
Default Primary Success Info Warning Danger Link
Default Primary Success Info Warning Danger Link
Default Primary Success Info Warning Danger Link

Tables

#	First Name	Last Name	Username
1	Mark	Otto	@mdo
2	Jacob	Thornton	@fat
3	Larry	the Bird	@twitter

#	First Name	Last Name	Username
1	Mark	Otto	@mdo
2	Jacob	Thornton	@fat
3	Larry	the Bird	@twitter

Basic Template

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="utf-8">
   <meta http-equiv="X-UA-Compatible" content="IE=edge">
   <meta name="viewport" content="width=device-width, initial-scale=1">
   <!-- The above 3 meta tags *must* come first in the head; any other head content must come *after* these tags -->
   <title>Bootstrap 101 Template</title>

   <!-- Bootstrap -->
   <link href="css/bootstrap.min.css" rel="stylesheet">

   <!– IE < 9 stuff erased… -->
</head>
<body>
<h1>Hello, world!</h1>

<!-- jQuery (necessary for Bootstrap's JavaScript plugins) -->
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js"></script>
<!-- Include all compiled plugins (below), or include individual files as needed -->
<script src="js/bootstrap.min.js"></script>
</body>
</html>
```

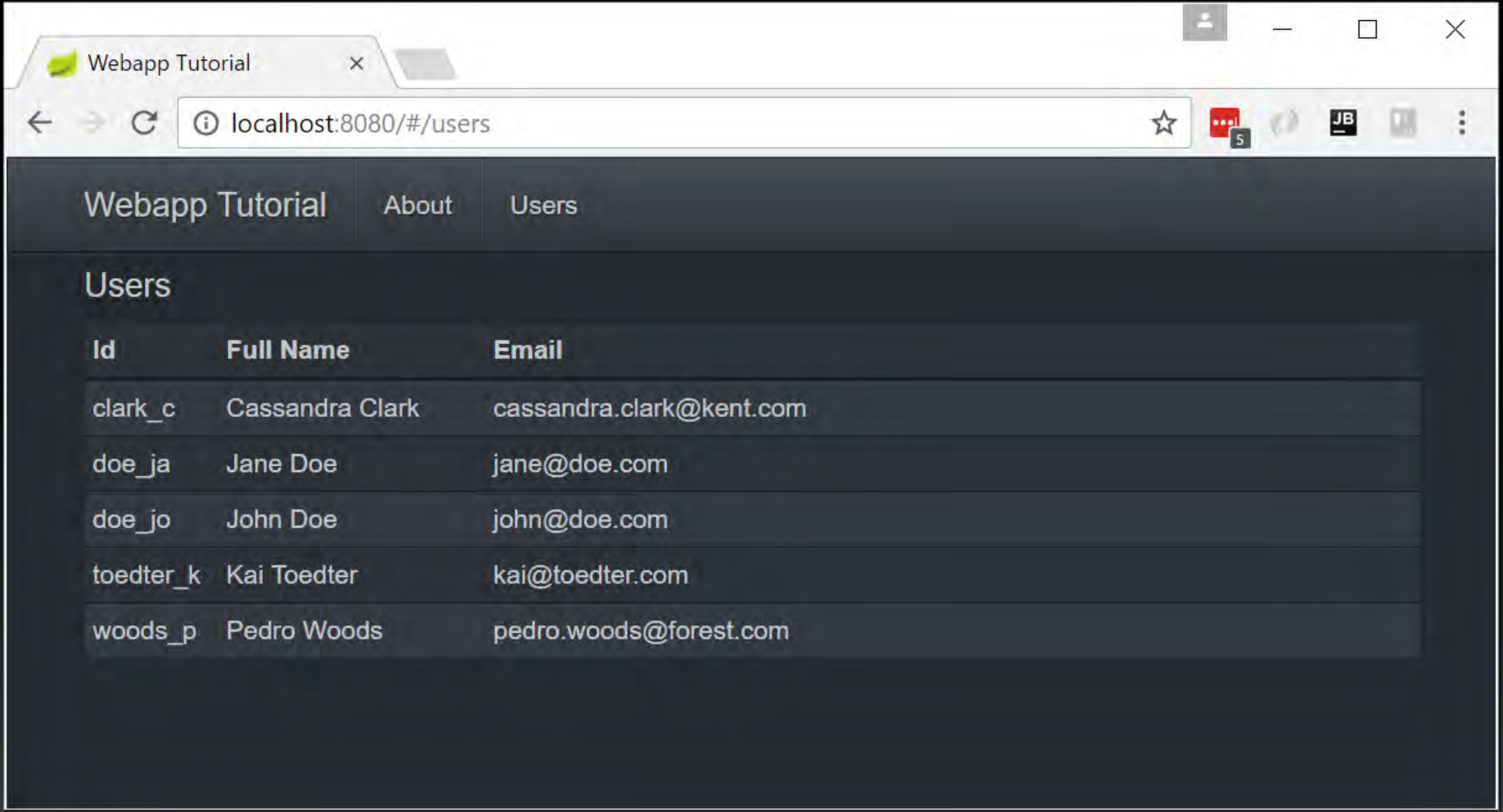
app.component.html

```html
<nav class="navbar navbar-default navbar-fixed-top">
  <div class="container">
    <div class="navbar-header">
      <a class="navbar-brand" href="#">Webapp Tutorial</a>
    </div>
    <div id="navbar" class="collapse navbar-collapse">
      <ul class="nav navbar-nav">
        <li><a [routerLink]="['about']">About</a></li>
        <li><a [routerLink]="['users']">Users</a></li>
      </ul>
    </div>
  </div>
</nav>
<router-outlet></router-outlet>
```

user.component.html

```
<div class="container">
  <h4>Users</h4>
  <table class="table table-striped table-condensed">
    …
```

Bootstrapped Tutorial Web App



Users

Id	Full Name	Email
clark_c	Cassandra Clark	cassandra.clark@kent.com
doe_ja	Jane Doe	jane@doe.com
doe_jo	John Doe	john@doe.com
toedter_k	Kai Toedter	kai@toedter.com
woods_p	Pedro Woods	pedro.woods@forest.com

Lab 6: Task 1

- Open terminal in lab6/complete
- Invoke: npm start
  - builds everything using webpack
  - Starts a Web server at port 3000
- Open http://localhost:3000 in a web browser
- You should see the web app with Bootstrap styling
- Play around with it

Lab 6: Task 2

- Work in lab6/initial
- Enhance index.html
  - Use slate.css for a dark Bootstrap theme
  - Add bootstrap css classes to
    - app.component.html
    - user.component.html
    - about.component.html

Putting it all together

Cross-Origin Resource Sharing (CORS)

From Wikipedia:

Cross-origin resource sharing (CORS) is a mechanism that enables many resources (e.g. fonts, JavaScript, etc.) on a web page to be requested from another domain outside the domain from which the resource originated.[

Spring CORS Filter

```
@Component
public class SimpleCORSFilter implements Filter {

  public void doFilter(ServletRequest req,
            ServletResponse res, FilterChain chain) throws IOException, ServletException {
    HttpServletResponse response = (HttpServletResponse) res;
    response.setHeader("Access-Control-Allow-Origin", "*");
    response.setHeader("Access-Control-Allow-Methods", "POST, PUT, PATCH, GET, OPTIONS, DELETE");
    response.setHeader("Access-Control-Max-Age", "3600");
    response.setHeader("Access-Control-Allow-Headers", "Origin, X-Requested-With, Content-Type, Accept");
    response.setHeader("Access-Control-Expose-Headers", "Location");
    chain.doFilter(req, res);
  }


  …
```

Angular HTTP

- The HTTP service uses Observables from rxjs
- The user service now makes an asynchronous call to the REST backend and returns an observable
- The user component subscribes for changes and updates its own view model

User Service with HTTP

```
@Injectable()
export class UsersService {
  constructor(private http: Http) {}

  public getUsers(): Observable<User[]> {
    let uri: string = 'http://localhost:8080/api/users';

  let observable: Observable<User[]> =
      this.http.get(uri)
        .map((response: Response) => response.json()._embedded['users'])
        .catch(this.handleError);

    return observable;
  }
…
```

UsersComponent with Subscription

```
export class UsersComponent {
  private users: User[];

  constructor(private usersService: UsersService) {}

  ngOnInit() {
    this.usersService.getUsers()
      .subscribe(
        (users: User[]) => this.users = users,
        error => console.error('UsersComponent: cannot get users'));
  }
}
```

Live Demo

Lab 7: Task 1

- Start the Spring Boot app from lab7
- Open terminal in lab7/complete
- Invoke: npm start
  - builds everything using webpack
  - Starts a Web server at port 3000
- Open http://localhost:3000 in a web browser
- You should see the web app
  - The user list is served by the backend service

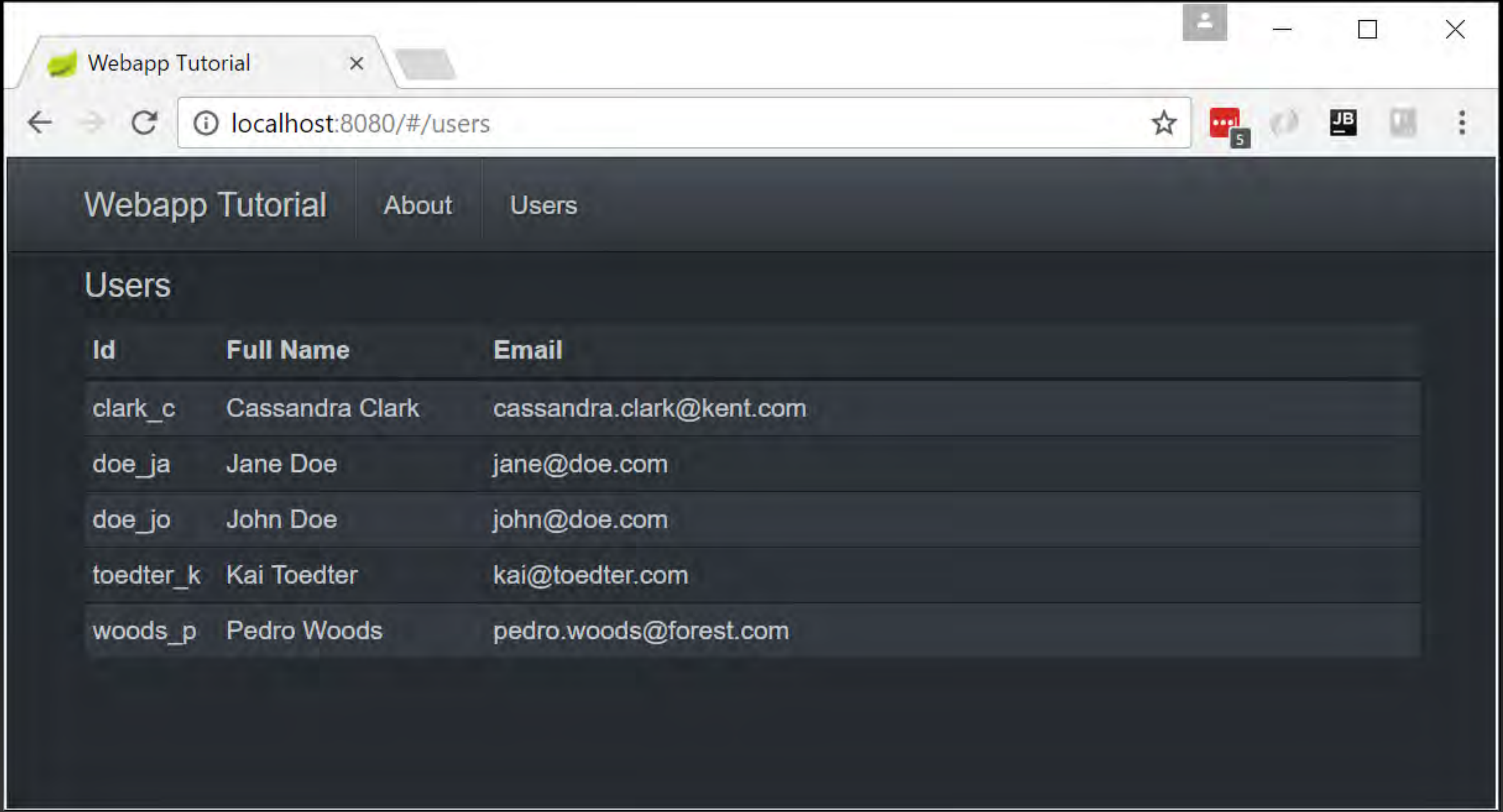
Lab 7: Task 2

- Work in lab7/initial
- Add a simple CORS filter to your Spring Boot app
    - Implement the doFilter method in SimpleCORSFilter.java

Lab 7: Task 3

- Implement the user service using HTTP
- Implement the users component with subscription

Final Application



Users

Id	Full Name	Email
clark_c	Cassandra Clark	cassandra.clark@kent.com
doe_ja	Jane Doe	jane@doe.com
doe_jo	John Doe	john@doe.com
toedter_k	Kai Toedter	kai@toedter.com
woods_p	Pedro Woods	pedro.woods@forest.com

Congratulations!

Discussion

License