JavaScript

Florian Bücklers & Hannes Kuhlmann

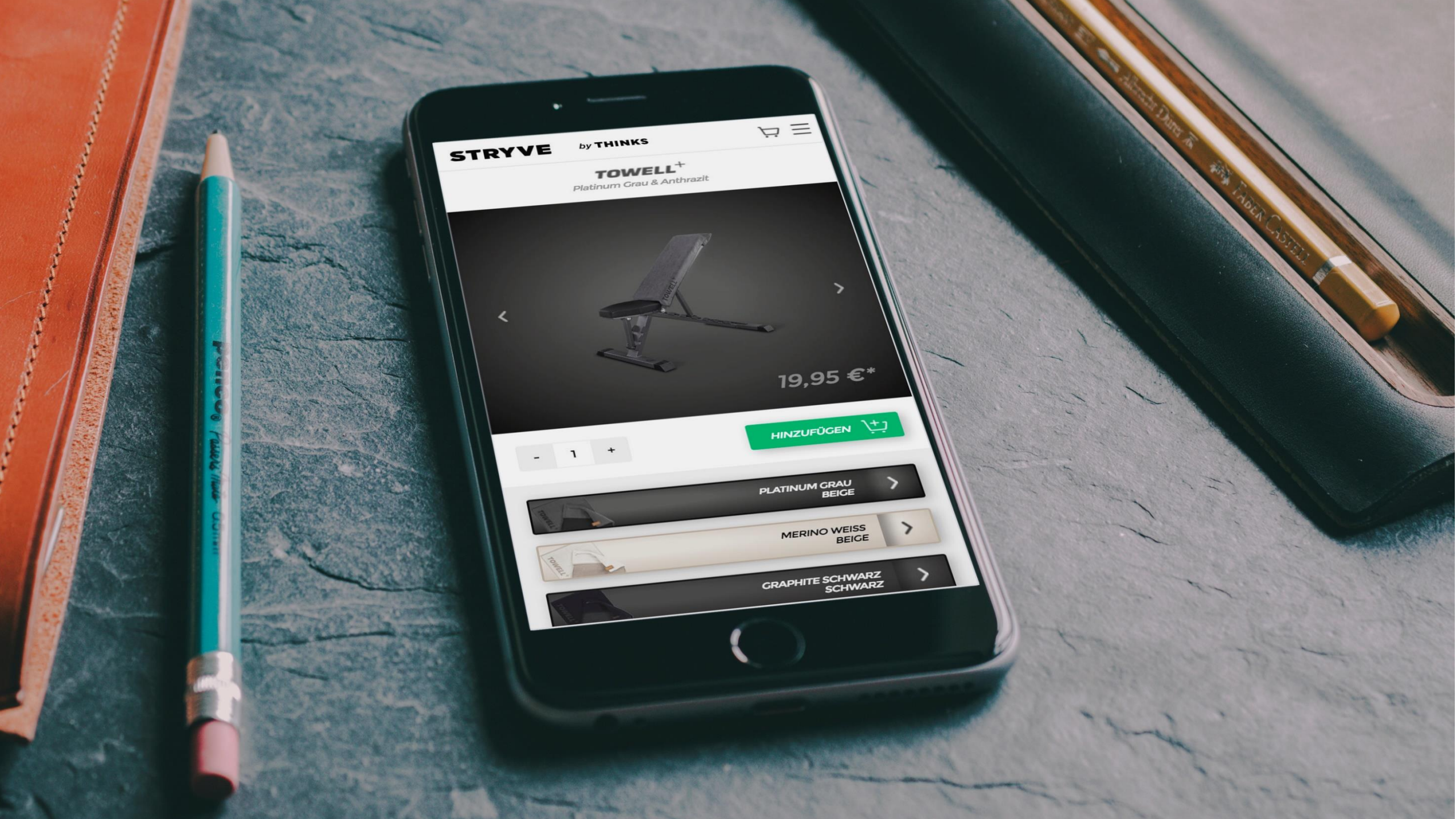
Angular2 – Real-Time-Anwendungen mit TypeScript entwickeln

Speaker **Q**uestion **R**ound

@baqendcom

Who we are

- Many years of experience in web dev
  - Both frontend and backend
  - Many Angular 1 projects
  - Very curious about Angular 2 → introduced it in a production web platform
- What we do: developing Baqend, a serverless backend for faster websites and apps
  - Great match for Angular2 (see our *Baqend+Angular 2* starter kit)


STRYVE
by THINKS
TOWELL+
Platinum Grau & Anthrazit
19,95 €*
HINZUFÜGEN
-
1
+
PLATINUM GRAU
BEIGE
MERINO WEISS
BEIGE
GRAPHITE SCHWARZ
SCHWARZ

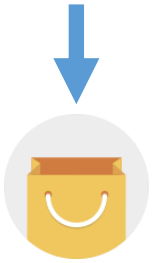
BaQend

STRYVE by THINKS Shop

< 1 second

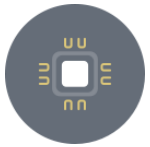
Page Load Time

Simultaneous Users



7.8%

Conversion Rate

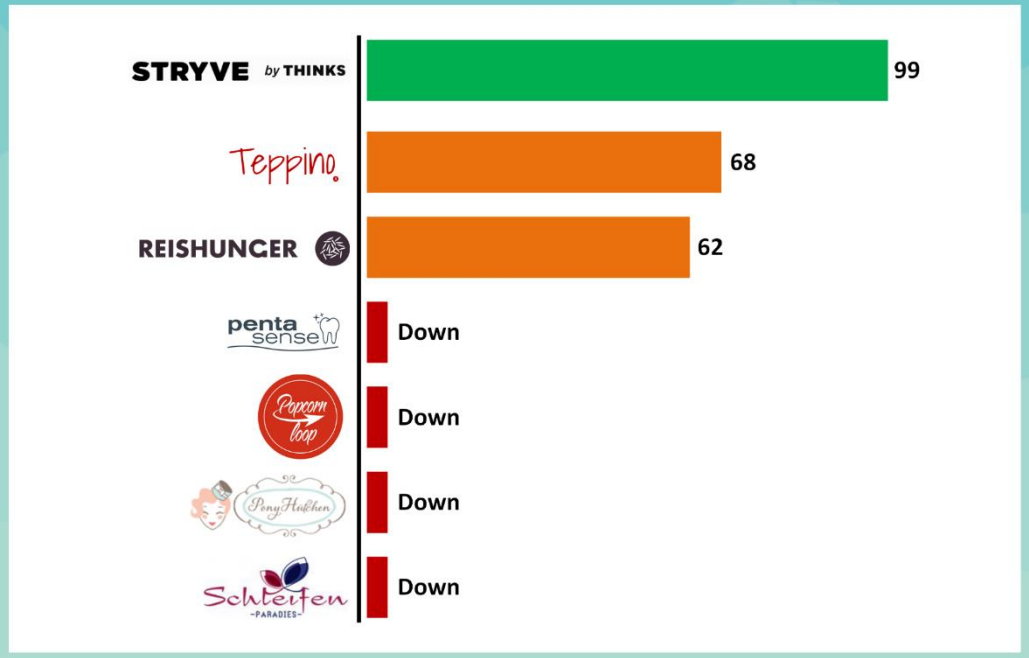


3%

Server Usage

Shops in "Die Höhle der Löwen"

The Google Page Speed Scores for Season 3, 09/06/2016

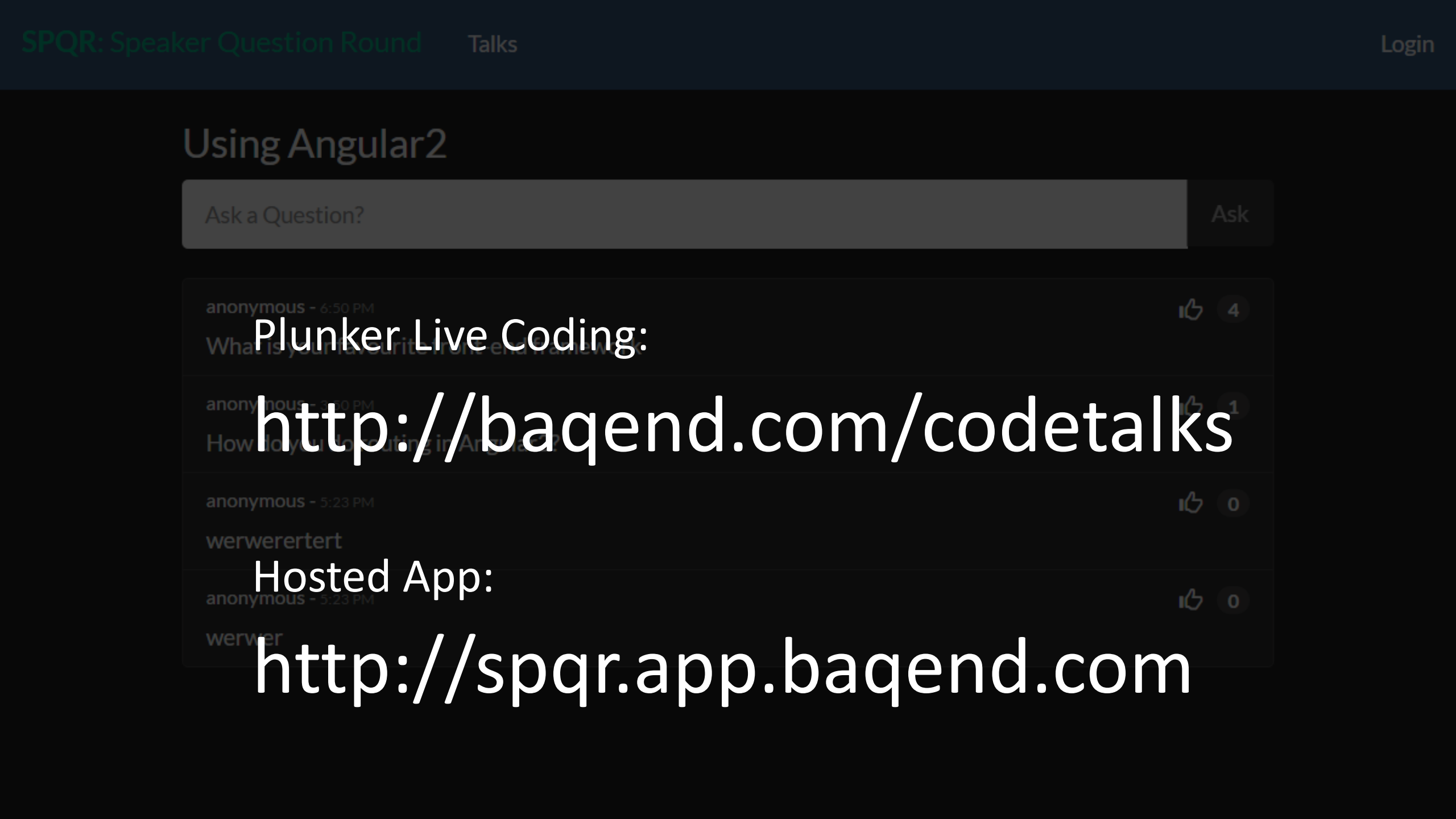


Angular 2

- A framework to build **client-side** applications
- Code can be written in Typescript, ES6, Dart or JavaScript (without transpiling)
- Has an expressive template language
- Powerful data binding
- Available as **release 2.0.0** since 14/09/2016

Why Angular 2?

- Fixes many performance pitfalls of Angular 1
- Modern design using ES6 features, e.g.:
  - Classes, Template strings
- Simplified API and clearer concepts
- Uses Typescript as the primary language for:
  - Interfaces
  - Type declarations
  - Decorators and Annotations


SPQR: Speaker Question Round
Talks
Login
Using Angular2
Ask a Question?
Ask
anonymous - 6:50 PM
4
anonymous - 5:23 PM
0
werwerertert
anonymous - 5:23 PM
0
werwer
Plunker Live Coding:
http://baqend.com/codetalks
Hosted App:
http://spqr.app.baqend.com

Components

- Components are the main classes where most of our code is
- **Template**:
  - Mostly HTML with some additional instructions
  - Styles are scoped to the component (similar to Web Components)
- **Component Class**:
  - Methods and properties that can be accessed and invoked from the rendered page
- Metadata specified in **Decorators:**
  - Additional instructions like styles, template and directives

Template expressions

- **Model to view** binding for values and properties

```
<span class="badge">{{question.votes}}</span>

<div [class.active]="question.state == 'active'">
```

- **View to model** binding

```
<div (click)="onClick($event)"></div>
```

- View to model & model to view (**2-way-binding**)

```
<input [(ngModel)]="value"></input>
<!-- short hand for -->
<input [ngModel]="value" (ngModel)="value = $event"></input>
```

Structural Directives *ngIf, *ngFor

- ***ngIf** conditionally renders a template

```
<!-- *ngIf paragraph -->
<div *ngIf="questions">
  We have some questions
</div>

<!-- [ngIf] with template -->
<template [ngIf]="questions ">
  <div>
    We have some questions
  </div>
</template>
```

- ***ngFor** loops over a collection

```
<div *ngFor="let question of questions">{{question.question}}</div>
```

Forms

- **ngModel** adds two-way binding between model and input value

```
<input type="text" [(ngModel)]="newQuestion" name="question" required>
```

- **ngSubmit** handles form submissions

```
<form (ngSubmit)="ask(newQuestion)">
```

- Access to the form controller **ngForm**

```
<form #questionForm="ngForm" (ngSubmit)="ask(questionForm.value)">
  …
  <button type="submit" [disabled]="!questionForm.valid">Ask</button>
</form>
```

Services

- Services are useful to share common code between different controllers.
- Services are **injectable**, so they can be used in any Component / Pipe / Directive …

```
@Injectable()
export class StorageService {
```

- Services are created by Angular when they are first requested and are treated as **singletons**

```
export class TalkComponent {
  //StorageService is injected to the component by angular
  constructor(private storageService:StorageService) {}
```

Pipes

- Pipes are template helpers to transform values

```
<small>{{question.date | date:'shortTime'}}</small>
```

- Pipes are *pure* by default
  - A pure pipe is only invoked if its primitive input changed or the reference of an object changed
- *Impure* Pipes
  - Needed if the pipe must be reevaluated when properties of an object changed or elements of an array are updated
  - Are always evaluated (expensive)

Router

- Routes are defined as **URL patterns** and handled by a target component

```
const routes: Routes = [
  { path: 'talk/:id', component: TalkComponent }
];
```

- Matching **route parameters** can be accesses by injecting the *ActivatedRoute*

```
export class TalkComponent {
  constructor(private route: ActivatedRoute) {}

  ngOnInit() {
    let id = this.route.snapshot.params['id'];
  }
}
```

Router Navigation

- The **routerLink** directive can be used to navigate to a another route

```
<a routerLink="/login">Login</a>
```

- The router can be used to navigate programmatically

```
constructor(private router: Router)

navigate(talk) {
  this.router.navigate(['/talk', talk.id]);
}
```

- Highlight active route links by setting a class

```
<a routerLink="/login" routerLinkActive="active">Login</a>
```

Route Subscription

- You can also **subscribe** to route parameter changes
  - Prevents recreating and redrawing the component when only a parameter changes

```
ngOnInit() {
  this.sub = this.route.params.subscribe(params => {
     let id = params['id'];
  });
}

ngOnDestroy() {
  this.sub.unsubscribe();
}
```

Directives

- Extend the behavior of HTML
  - Most directives introduce new attributes or HTML elements
  - The controller can be exposed with *exportAs*

```
@Directive({
  selector: '[collapse]',
  exportAs: 'Collapse'
})
export class CollapseDirective {
```

- Controller methods can be used within the template

```
<button type="button" (click)="navbar.toggle()"></button>
<div collapse #navbar="Collapse">
```

Model to directive binding

- A directive has no direct access to outer scope
  - Instead model data can bind to a directive

```
<div [collapse]="navbarCollapse">
```

  - The directive subscribes to changes

```
export class CollapseDirective {
  @Input() collapse:boolean;

  ngOnChanges(changes: {[propKey: string]: SimpleChange}) {
    if (changes.collapse) {
      //changed by model
      let from = changes.collapse.previousValue;
      let to = changes.collapse.currentValue;
    }
  }
}
```

Directive to model binding

- A directive has no direct access to the outer scope
  - Data can be sent to the model

```
export class CollapseDirective {
  @Output() collapse = new EventEmitter<boolean>();

  toggle() {
    this.expanded = !this.expanded;
    this.collapse.emit(this.expanded);
  }
}
```

  - And be subscribed to in the view

```
<div (collapse)="navbarCollapse = $event">
```

Directive <-> model binding

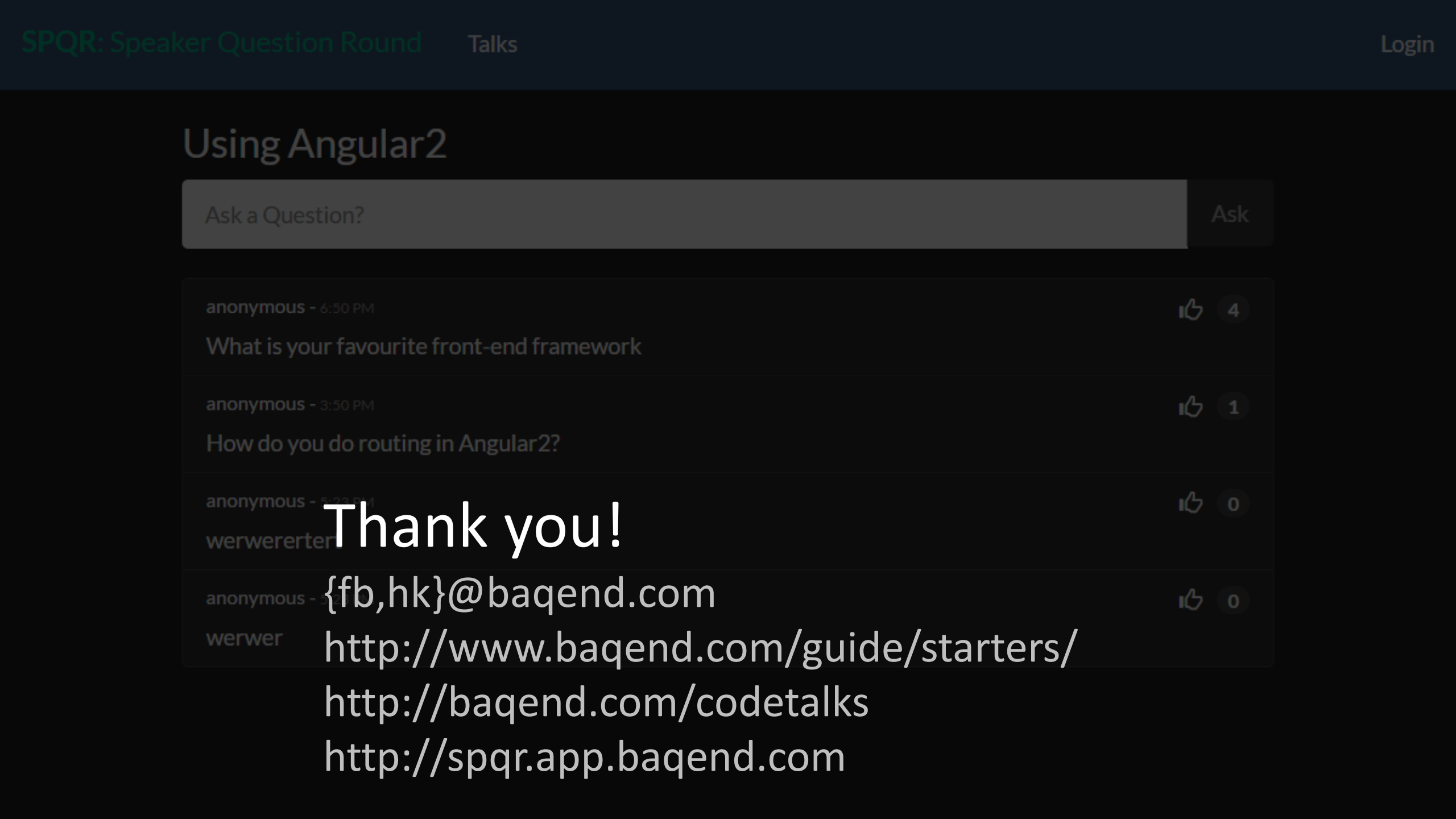
- Binding can be two-way, similar to components:

```
<div [(collapse)]="navbarCollapse">

export class CollapseDirective {
  @Input() collapse:boolean;
  @Output() collapseChange = new EventEmitter<boolean>();

  ngOnChanges(changes: {[propKey: string]: SimpleChange}) {
    if (changes.collapse)
      //changed by model
  }

  toggle() {
    this.collapseChange.emit(!this.expanded);
  }
}
```



Thank you!

{fb,hk}@baqend.com

http://www.baqend.com/guide/starters/

http://baqend.com/codetalks

http://spqr.app.baqend.com