

Safety First: Wie eine Angular Anwendung abgesichert wird

Talk – Track 2

Hannah Schieber

Corina Hampel

Hochschule Aalen

Wer sind wir? – Corina Hampel



Ausbildung

Fachinformatikerin für Anwendungsentwicklung
(duale Ausbildung)
Bachelor Informatik, Schwerpunkt IT-Sicherheit
an der HS Aalen



Aktuell

Wissenschaftliche Mitarbeiterin HS Aalen

- Vorlesung Einführung IT-Sicherheit
- Vorlesung Sichere Programmierung

Studium Master Informatik
Schwerpunkt IT-Sicherheit, HS Aalen

Wer sind wir? – Hannah Schieber



Ausbildung

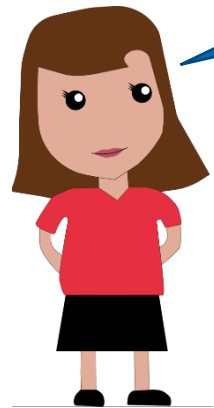
Bachelor Informatik an der HS Aalen



Aktuell

Dualer Master Informatik @Audi,
Schwerpunkt Safety and Security an der TH Ingolstadt

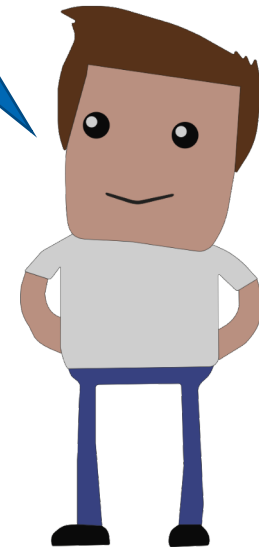
Startup: [Wastelesslife.info](https://wastelesslife.info)



Bloggerin

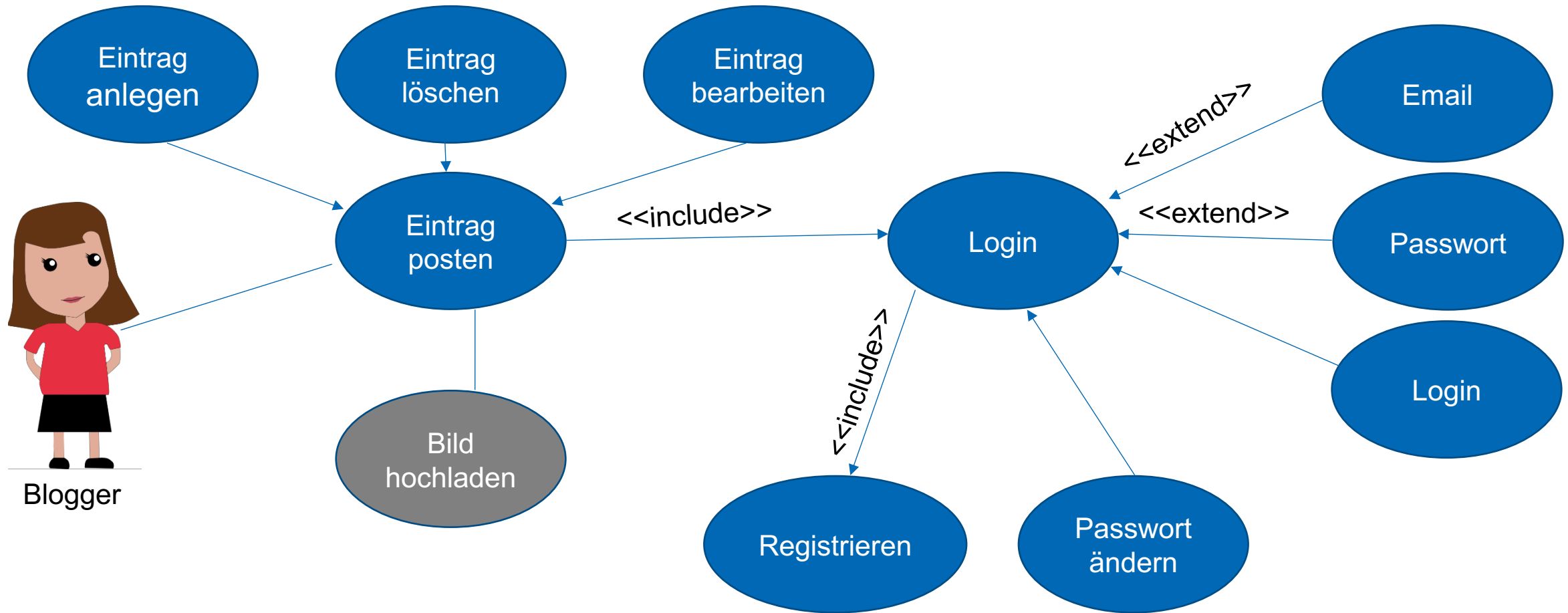
Ich will
einen Blog!

Was soll er
können?



Entwickler

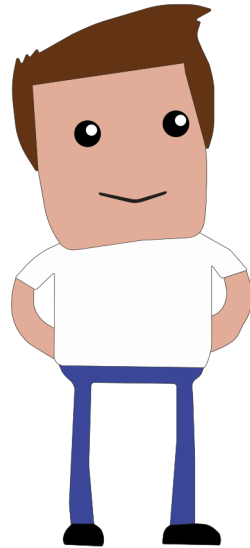
Use Case



Security by Design – Prinzipien zur Entwicklung sicherer Software

- Angriffsfläche minimieren
- Sichere Defaults
- Minimale Rechte vergeben
- Auch ein Admin braucht nicht alle Rechte, Trennung von Verantwortlichkeiten
- Vertrauen ist gut, Kontrolle ist besser → Kein Vertrauen in nur eine Kontrolle
- Ausgiebige Fehlerbehandlungen
- Man vertraut keinen Fremden(-Services)
- Kein Security by Obscurity
- Mut zur (Code-)Schlichtheit
- Gründliche Behebung bei Sicherheitsvorfällen

Zusammengefasst:



Trust no one!!!

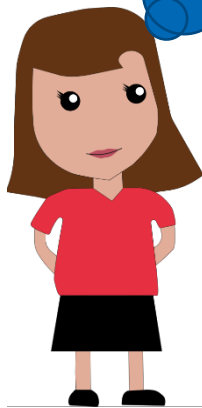
Was sind denn die kritischsten Sicherheitslücken?



OWASP Top 10 v2017

1. Injection
2. Authentifizierungsfehler
3. Verlust der Vertraulichkeit sensibler Daten
4. XML External Entities (XXE)
5. Fehler in der Zugriffskontrolle
6. Sicherheitsrelevante Fehlkonfiguration
7. Cross-Site Scripting (XSS)
8. Unsichere Deserialisierung
9. Nutzung von Komponenten mit bekannten Schwachstellen
10. Unzureichendes Logging & Monitoring.

Muss ich mich um
das alles kümmern?

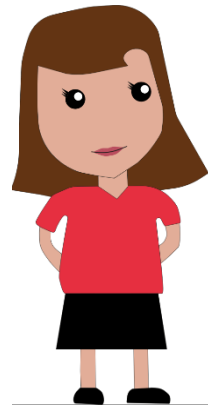


OWASP Top 10

1. Injection
2. Authentifizierungsfehler
3. Verlust der Vertraulichkeit sensibler Daten
4. XML External Entities (XXE)
5. Fehler in der Zugriffskontrolle
6. Sicherheitsrelevante Fehlkonfiguration
7. Cross-Site Scripting (XSS)
8. Unsichere Deserialisierung
9. Nutzung von Komponenten mit bekannten Schwachstellen
10. Unzureichendes Logging & Monitoring.



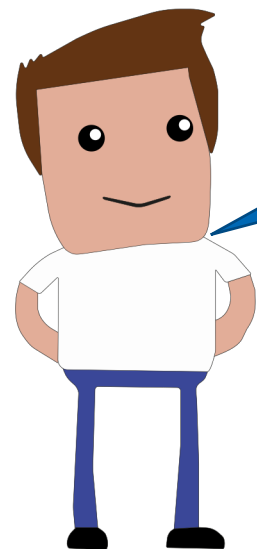
Grundsätzlich



Wir verwenden
keine alten
Versionen!

Den Server richtig konfigurieren

- Keine unnötigen Informationen preisgeben
- Hilfreiche Module, z.B. libapache2-mod-evasive für Apache
- HTTPS erzwingen
- Unsichere SSL/TLS Versionen blockieren
(Mindestanf. TLS 1.2, aktuell TLS 1.3)
- CipherSuites vorgeben
- SSL-Zertifikat einrichten
- Logs einschalten
- Zum Prüfen: kurzer Check mit Tools wie Nikto, dirb, nmap,... → **Erlaubnis erfragen!**



Nie auf
Standardconfigs
verlassen!

OWASP Top 10

1. Injection
2. Authentifizierungsfehler
3. Verlust der Vertraulichkeit sensibler Daten
4.
5. Fehler in der Zugriffskontrolle
6.
7. Cross-Site Scripting (XSS)
8. Unsichere Deserialisierung
9. Nutzung von Komponenten mit bekannten Schwachstellen



Nutzung von Komponenten mit bekannten Schwachstellen

- Keine nicht-verifizierten Add-ons/Extensions nutzen.
- Fertig

OWASP Top 10

1. Injection
2. Authentifizierungsfehler
3. Verlust der Vertraulichkeit sensibler Daten
5. Fehler in der Zugriffskontrolle
6. Sicherheitsrelevante Fehlkonfiguration
8. Unsichere Deserialisierung

Auftritt Angular:

JWT-Tokens
Route Guards

Was bedeuten diese Risiken für Lisa?

- Wie wird die Anwendung abgesichert?
 - Zwei Seiten die beachtet werden müssen:

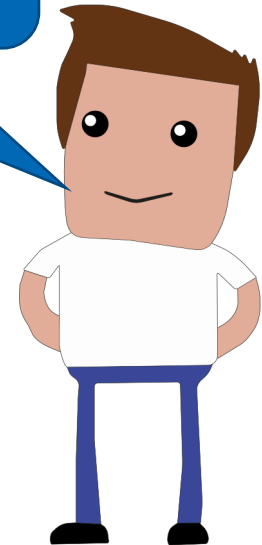


Client



Server

Die Webanwendung muss abgesichert werden!



Anwendung

Backend

Python
SQLite



Austausch

JSON Web Token
Rest - Anfragen



{ REST }

Frontend

Angular
Bootstrap



Angular



Allgemein:

Front-End-Webapplikationsframework

TypeScript basiert

Hierarchie von Komponenten



Vorreiter:

AngularJS



Aktuell:

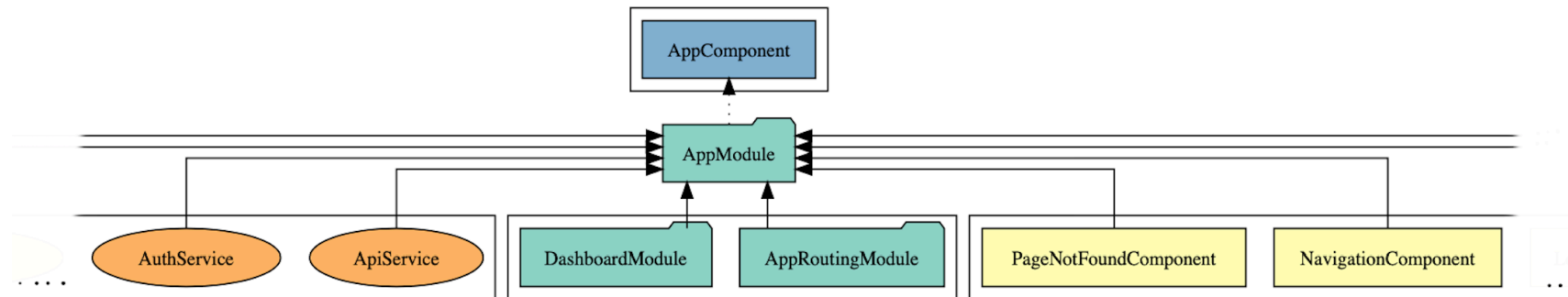
Angular 2+

Cross – Plattform mit Ionic



Angular

- Aufbau:
 - Module
 - deklariert einen Kompilierungskontext für eine Reihe von Komponenten
 - Component
 - definiert eine Klasse
 - enthält Anwendungsdaten und -logik
- Service
 - stellt Daten/Logik bereit, die von verschiedenen Klassen genutzt werden
 - Beispiele:
 - AuthService
 - ApiService

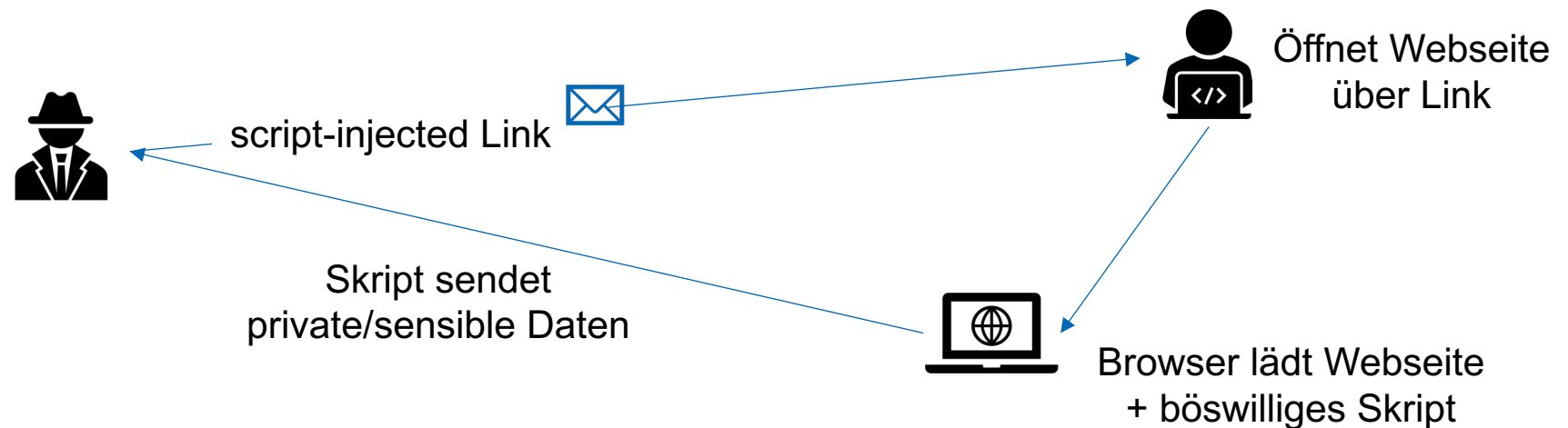


Absicherung der Anwendung

- Angular - Sicherheitscheck
- Angular Route Guards
- JSON Web Token
- Hosting auf Server mit SSL Zertifikat

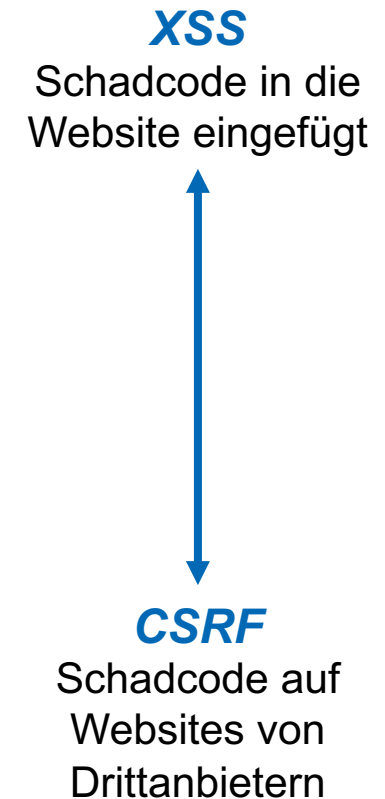
Angular - Sicherheitscheck

- *Built – In Protections*
 - Verhindern von Cross – Site – Scripting (XSS)
 - behandelt alle Werte standardmäßig als „**untrusted**“
 - „**untrusted**“ – Values → bereinigt und ausgeblendet
 - **keine** kontrollierbaren Templates verwenden
 - User Input und Angular Template niemals verbinden
 - offline template compiler verwenden → Verhindern von Template Injection



Angular - Sicherheitscheck

- *Cross – site script inclusion (XSSI)*
 - Überschreiben natives JS-Objektconstructoren
 - Anschließend API-URL mit einem <script> -Tag einschließen
 - Erfolgreich wenn: zurückgegebenes JSON als JavaScript ausführbar ist
 - Angular HttpClient – Bibliothek → escapen von: `"})} ', \ n"`
- *Cross – Site Request Forgery*
 - Bekannte Anti-XSRF Technik:
 - Anwendung sendet „Random – Auth“ – Cookie
 - Client fügt Cookie an Request Header an
 - Server prüft erhaltenen Cookie
 - Angular HttpClient – Bibliothek unterstützt die Clientseitige Implementierung



Angular - Sicherheitscheck

- *Sanitization*
 - Inspektion von „**untrusted**“ Values
 - Werte, die in CSS ungefährlich sind können in einer URL gefährlich sein
- Security Context:
 - HTML
 - Binding an **innerHTML**
 - Style
 - Binding von CSS an **Style** - Property
 - URL
 - URL – Properties, z.B. **<a href>**
 - Resource URL
 - Laden von „entferntem Code“, z.B. **<script src>**

Angular - Sicherheitscheck

- Content Security Policy (CSP)
 - defense-in-depth Technik zur Vermeidung von XSS
 - Clientseitig
 - Setzen eines Metas im HTML Header
 - Serverseitig
 - Setzen des HTTP Headers
- DomSanitizer
 - Definieren von Ausnahmen
 - `bypassSecurityTrustHtml`
 - `bypassSecurityTrustScript`
 - `bypassSecurityTrustStyle`
 - `bypassSecurityTrustUrl`
 - `bypassSecurityTrustResourceUrl`



```
<h4>An untrusted URL:</h4>
<p><a [href]="dangerousUrl">Click me</a></p>
<h4>A trusted URL:</h4>
<p><a [href]="trustedUrl">Click me</a></p>
```

```
constructor(private sanitizer: DomSanitizer) {
  // javascript: URLs are dangerous if attacker controlled.
  // Angular sanitizes them in data binding, but you can
  // explicitly tell Angular to trust this value:
  this.dangerousUrl = 'javascript:alert("Hi there")';
  this.trustedUrl = sanitizer.bypassSecurityTrustUrl(this.dangerousUrl);
}
```

Angular - Sicherheitscheck

- Beinhaltet nicht:
 - Authentifizierung (Wer ist der User?)
 - Autorisierung (Was darf der User?)
- Markiert risikoreiche Bibliotheken als „Security - Risk“ → vermeiden!
- Best Practice
 - Aktuellste Version verwenden
 - Regelmäßige Wartung
 - Standardbibliotheken nutzen

Was sind Route Guards?

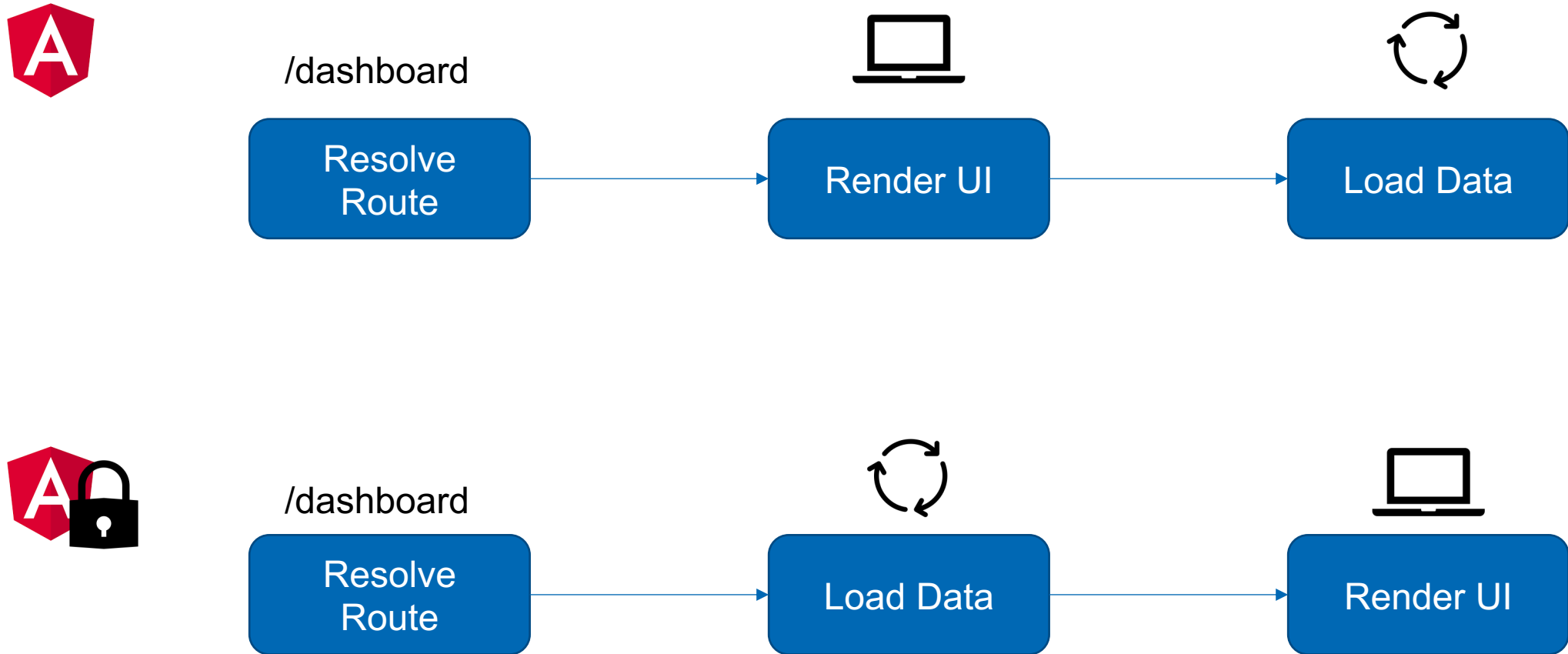
- Angular Router
 - Grundsätzlich Navigation zwischen den Seiten
- Angular Route Guard
 - Festlegung welche Routen zugänglich sind
 - Guard – Schnittstelle liefert True oder False
- Fünf unterschiedliche Arten von Guards:
 - CanActivate
 - CanActivateChild
 - CanDeactivate
 - CanLoad
 - Resolve

```
@Injectable()
export class AuthGuardService implements CanActivate {

  constructor(private authService: AuthService, private router: Router) {
  }

  canActivate(next: ActivatedRouteSnapshot, state: RouterStateSnapshot):
  Observable<boolean> | Promise<boolean> | boolean {
    if (this.authService.isAuthenticated()) {
      return true;
    }
    // navigate to login page
    this.router.navigate(['login']);
    // safely redirect
    return false;
  }
}
```

Route Guards



JSON Web Token

- Genormter Access – Token
- Token nicht in Cookie speichern um CSRF zu verhindern

„JSON Web Tokens are an open, industry standard RFC 7519 method for representing claims securely between two parties.“

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWwiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.cThlloDvwdueQB468K5xDc5633seEFogwxjF_xSJyQQ

JSON Web Token

Header

Payload

Verified
Signature

```
{  
  "alg": "HS256",  
  "typ": "JWT"  
}  
  
{  
  "sub": "1234567890",  
  "name": "Jane Doe",  
  "iat": 1516239022  
}  
  
HMACSHA256 (  
  base64UrlEncode(header) + "." +  
  base64UrlEncode(payload),  
  your-256-bit-secret  
)
```

SQL - Injection

- Ausnutzen von Sicherheitslücken bei der Verwendung von SQL – Datenbanken

**http://localhost:5000/getallentriesfromuser/1 UNION
ALL SELECT *, NULL, NULL FROM users**

```
[  
  {  
    "id": 1,  
    "title": "test",  
    "text": "test",  
    "date": "test@test.de",  
    "author": "07947940510",  
    "author_id": null,  
    "tags": null  
  },  
  {  
    ...  
  }  
]
```

/changeuserpw

```
{  
  "newpw": "test"  
}
```

Wie erreichen wir unsere OWASP Ziele?

1. Injection → Input Type setzen
2. Authentifizierungsfehler → AuthGuards
3. Verlust der Vertraulichkeit sensibler Daten
4. XML External Entities (XXE) → erschwert durch AuthGuards
5. Fehler in der Zugriffskontrolle → richtige Konfiguration AuthGuards
6. Sicherheitsrelevante Fehlkonfiguration → Einsatz AuthGuards, Deprecated Libraries vermeiden
7. Cross-Site Scripting (XSS) → Angular Sicherheitscheck
8. Unsichere Deserialisierung → nicht vertrauenswürdige Quellen vermeiden
9. Nutzung von Komponenten mit bekannten Schwachstellen
10. Unzureichendes Logging & Monitoring → Backend richtig konfigurieren

Danke für Eure Aufmerksamkeit!

- <https://material.angular.io/guide/getting-started>
- <https://worldvectorlogo.com/downloaded/jwtio-json-web-token>
- https://de.m.wikipedia.org/wiki/Datei:Bootstrap_logo.svg
- <https://glear.de/2019/sqlitenews/>
- https://www.ictshore.com/software-design/rest-architecture/attachment/sfw0002-01-rest_architecture/
- <https://ionicacademy.com/ionic-logo-portrait/>
- https://medium.com/@ryanchenkie_40935/angular-authentication-using-route-guards-bf7a4ca13ae3