TECHNOLOGY DAY · 12. NOVEMBER 2026

Passkeys integrieren – Teil 2

Von der Demo zum echten System

INNOQ



FELIX SCHUMACHER
SENIOR CONSULTANT

Agenda

- Rückblick auf Teil 1
- Was ich letztes Mal gebaut habe
- Schritt für Schritt zum echten System
- Setup mit IdP: Keycloak
- Migration: von Passwörtern zu Passkeys
- Ausblick: iPhone, YubiKey, native Apps



Kapitel 1

Rückblick auf Teil 1



Wer prüft schon *jede URL*?

Das Problem: Phishing

- Passwörter werden geleakt, wiederverwendet, erraten
- Eine Login-Seite zu klonen dauert Minuten (Social Engineering Toolkit)
- Mit KI sehen Phishing-Mails und Fake-Seiten immer echter aus

Passkeys in einer Folie

- Statt eines Passworts: ein **Schlüsselpaar** pro Website
- Der **Public Key** liegt beim Server, der **Private Key** verlässt das Gerät nicht
- Login = Challenge vom Server, signiert mit dem Private Key
- Die **Domain ist Teil des Schlüssels** → auf `paypa1.com` gibt es keinen Passkey
- Standard: W3C WebAuthn, „Passkey“ ist der landläufige Name

Drei Wege, die ich ausprobiert habe

	WEG	ERGEBNIS
✗	ChatGPT + Yubico-Library	Dependency-Hölle, kompiliert nicht
✗	ChatGPT: „Implementiere WebAuthn mit Spring“	Spec von Hand implementiert, funktioniert nicht
✓	**Spring Security WebAuthn**	wenige Zeilen, läuft – viel Magie
✓	Corbado (Passkeys as a Service)	am einfachsten – aber Abhängigkeit

Ein Jahr später: nochmal probiert

	WEG	DAMALS	HEUTE
	KI + Yubico-Library	kompiliert nicht	läuft, 1. Anlauf – ~580 Zeilen, 6 Klassen
	KI: „Implementiere WebAuthn mit Spring“	Spec von Hand, kaputt	läuft – nimmt jetzt Spring Security, ~90 Zeilen
	Spring Security WebAuthn	Snapshot, viel Magie	stabil (Boot 4.1.2), weiter wenige Zeilen
	Corbado	am einfachsten	Enterprise-Layer auf vorhandenem IdP, Java-SDK archiviert

Was offen geblieben ist

- Ein Passkey pro Gerät? Mehrere Geräte pro Konto?
- Sync über die Cloud – und damit wieder ein Angriffsweg?
- Recovery: Gerät weg, Konto weg?
- Transition von Passwörtern zu Passkeys?
- Passkey als zweiter Faktor, Step-up, Keycloak? (Fragen aus dem Publikum)

... und vielleicht kommt dann auch noch mal ein tiefergehender Vortrag, wo ich es dann mal richtig in voller Tiefe implementiere, sodass ich auch sagen würde, dass es vielleicht produktionsreif ist.

— **Ich, Teil 1** (Minute 39)

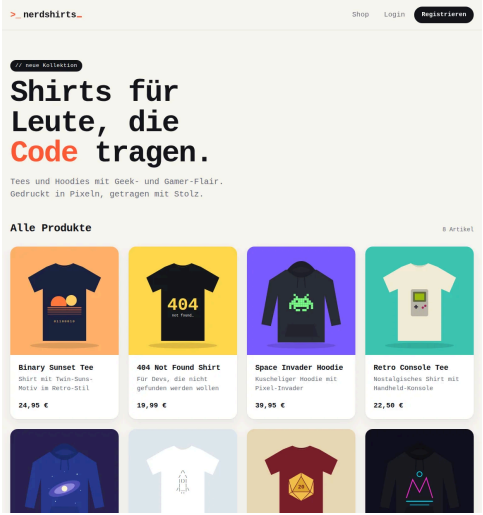


Kapitel 2

Was ich letztes Mal gebaut habe

Der Nerdshirts-Shop

- Spring Boot, Thymeleaf, JPA, Postgres
- Login mit E-Mail und Passwort
- von Codex generiert (das hat funktioniert)
- Repo: `ch4inl3ss/demo-webshop`



Der ganze Diff aus Teil 1

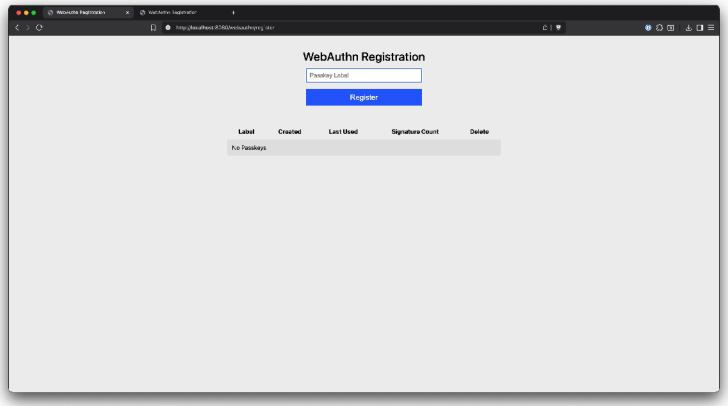
```
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-webauthn</artifactId>
</dependency>
```

```
return http.authorizeHttpRequests(ht -> ht.anyRequest().authenticated())
        .formLogin(withDefaults())
        .webAuthn(webauth -> webauth
                .allowedOrigins("http://localhost:8080")
                .rpId("localhost")
                .rpName("webshopname")
        )
        .build();
```

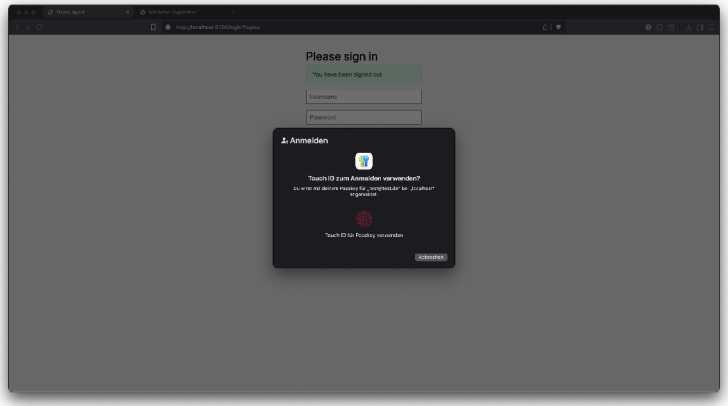
Live-Demo



Branch `step-0-aufraeumen`



`/webauthn/register` – von Spring generiert



Login per Touch ID

Was Spring hier alles für mich tut

WAS	WO
Registrierungs-Seite inkl. Passkey-Liste	`GET /webauthn/register`
Optionen für die Registrierung	`POST /webauthn/register/options`
Registrierung abschließen	`POST /webauthn/register`
Optionen für den Login	`POST /webauthn/authenticate/options`
Login	`POST /login/webauthn`
Login-Seite mit Passkey-Button + JavaScript	generiert



Und jetzt:

App neu starten.

Warum das noch nicht produktionsreif ist

- Passkeys liegen **nur im Arbeitsspeicher**
- Die Shop-Security ist plattgemacht: alles braucht Login, eigene Login-Seite weg
- Origin und RP-ID fest auf `localhost`, kein HTTPS
- Passkey-Verwaltung nur über eine generierte Seite
- Spring Boot `4.0.0-SNAPSHOT`

Kapitel 3

Schritt für Schritt zum echten System

Der Plan

	SCHRITT	BRANCH	LÖST
	0	`step-0-aufraeumen`	Snapshot-Version, `target/` im Repo
	1	`step-1-persistenz`	Passkeys überleben den Neustart
	2	`step-2-security`	Shop-Security und eigene Login-Seite zurück
	3	`step-3-verwaltung`	Mehrere Passkeys pro Konto, eigene UI
	4	`step-4-domain`	HTTPS, echte Domain, Konfiguration
	5	`step-5-step-up`	Admin-Bereich mit erneuter Bestätigung

🧹 Schritt 0: Aufräumen

- 🧪 Spring Boot `4.0.0-SNAPSHOT` → `4.1.2`, Snapshot-Repos raus
- 🗑️ `target/` und `.idea/` aus dem Repo (das Jar hatte 66 MB)
- 📦 WebAuthn bleibt eine einzige Dependency: `spring-security-webauthn`

Schritt 1: Passkeys persistieren

Spring bringt JDBC-Implementierungen für beide Repositories mit:

```
@Bean
PublicKeyCredentialUserEntityRepository userEntities(JdbcOperations jdbc) {
    return new JdbcPublicKeyCredentialUserEntityRepository(jdbc);
}

@Bean
UserCredentialRepository userCredentials(JdbcOperations jdbc) {
    return new JdbcUserCredentialRepository(jdbc);
}
```

Dazu zwei Tabellen: `user_entities` und `user_credentials` (Schema liefert Spring Security mit).

Was in der Datenbank landet

USER_ENTITIES	USER_CREDENTIALS
id – zufällige User-Handle (nicht die E-Mail!)	credential_id
name – Benutzername	user_entity_user_id
display_name	public_key
	signature_count , label , created , last_used
	backup_eligible , backup_state

Live-Demo



Branch `step-1-persistenz`

Schritt 2: Die Shop-Security zurückholen

```
http
    .authorizeHttpRequests(auth -> auth
        .requestMatchers("/", "/products/**", "/register", "/css/**", "/js/**", "/images/**").permitAll()
        .requestMatchers("/admin/**").hasRole("ADMIN")
        .anyRequest().authenticated())
    .formLogin(form -> form.loginPage("/login").permitAll())
    .webAuthn(webAuthn -> webAuthn
        .rpName("Nerdshirts")
        .rpId("localhost")
        .allowedOrigins("http://localhost:8080"))
    .logout(logout -> logout.logoutSuccessUrl("/login?logout").permitAll())
```

Eigene Login-Seite mit Passkey-Button

```
const res = await fetch("/webauthn/authenticate/options", { method: "POST", headers: csrf });
const options = PublicKeyCredential.parseRequestOptionsFromJSON(await res.json());

const credential = await navigator.credentials.get({ publicKey: options });
// → Touch ID / Face ID / YubiKey

const login = await fetch("/login/webauthn", {
  method: "POST",
  headers: { ...csrf, "Content-Type": "application/json" },
  body: JSON.stringify({ ...credential.toJSON(), credType: credential.type }),
});
location.href = (await login.json()).redirectUrl;
```

|** Passwort-Login bleibt als Fallback auf derselben Seite

🔑 Schritt 3: Mehrere Passkeys pro Konto



MacBook

Touch ID, iCloud-Schlüsselbund



iPhone

Face ID, synchronisiert



YubiKey

gerätegebunden, Backup-Schlüssel



Weiteres Gerät

jederzeit hinzufügen

- Liste mit Label und „zuletzt benutzt“ in der eigenen Profilseite (`/webauthn/register` kann das generiert schon)
- ⚠ Den letzten Login-Weg nicht löschen lassen

🔔 Den Browser auf dem Laufenden halten

Gelöschter Passkey auf dem Server – aber der Passwortmanager bietet ihn weiter an?

```
PublicKeyCredential.signalAllAcceptedCredentials({
  rpId: "nerdshirts.example",
  userId: userHandle,              // base64url
  allAcceptedCredentialIds: ids,   // was der Server noch kennt
});
```

- WebAuthn **Signal API**: Server teilt dem Passkey-Manager mit, was noch gültig ist
- Auch für geänderte Namen: `signalCurrentUserDetails(...)`

🌐 Schritt 4: Echte Domain, echtes HTTPS

- WebAuthn braucht einen **Secure Context**: HTTPS (nur `localhost` ist ausgenommen)
- `rpId` = die Domain, an die die Passkeys gebunden sind
- `allowedOrigins` = alle Origins, von denen Logins kommen dürfen
- Beides gehört in die Konfiguration, nicht in den Code



Wichtigste Entscheidung des ganzen Vortrags:

Die RP-ID nachträglich zu ändern, macht alle Passkeys ungültig.

Schritt 5: Step-up für kritische Bereiche

- Frage aus Teil 1: Braucht man noch einen zweiten Faktor zum Passkey?
- Normaler Login: **Passkey reicht** (Besitz + Biometrie/PIN = zwei Faktoren)
- Admin-Bereich oder Bestellung abschicken: **erneut bestätigen lassen**
- `userVerification: required` erzwingt Biometrie/PIN statt nur „Gerät vorhanden“

Live-Demo



Branch `step-5-step-up`

Kapitel 4

Setup mit IdP: Keycloak

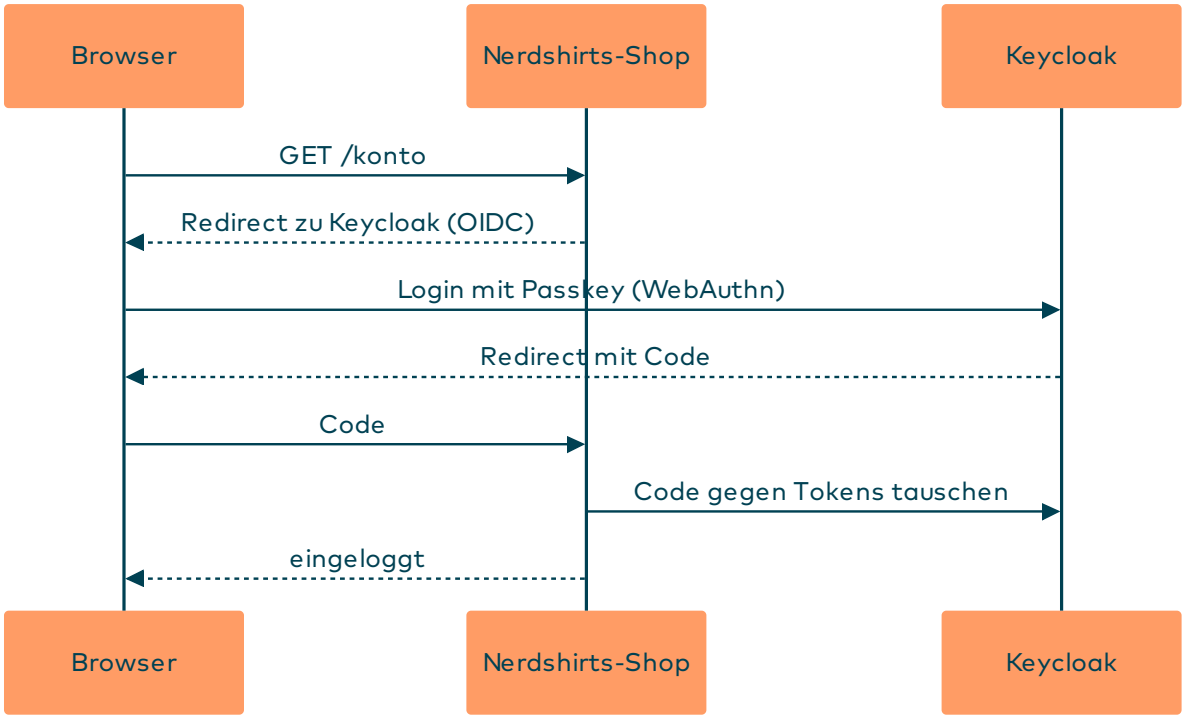


Frage aus Teil 1: *„Wie schwer ist das in Keycloak?“*

Warum überhaupt ein IdP?

- Mehrere Anwendungen, ein Login – Passkeys nur an einer Stelle
- Der Shop kennt keine Passwörter und keine Passkeys mehr
- Recovery, MFA, Step-up, Sperren: alles zentral
- Keycloak: Open Source, selbst gehostet

Architektur



Keycloak dazustellen

```yaml
keycloak:
  image: keycloak/keycloak:26.8.2
  command: start-dev --import-realm
  environment:
    KC_BOOTSTRAP_ADMIN_USERNAME: admin
    KC_BOOTSTRAP_ADMIN_PASSWORD: admin
  ports:
    - "8180:8080"
  volumes:
    - keycloak-data:/opt/keycloak/data
    - ./keycloak/nerdshirts-realm.json:/opt/keycloak/data/import/nerdshirts-realm.json:ro
```

Realm, Client, Rollen und Test-Nutzer als JSON im Repo – reproduzierbar statt durchgeklickt

Den Shop auf OIDC umstellen

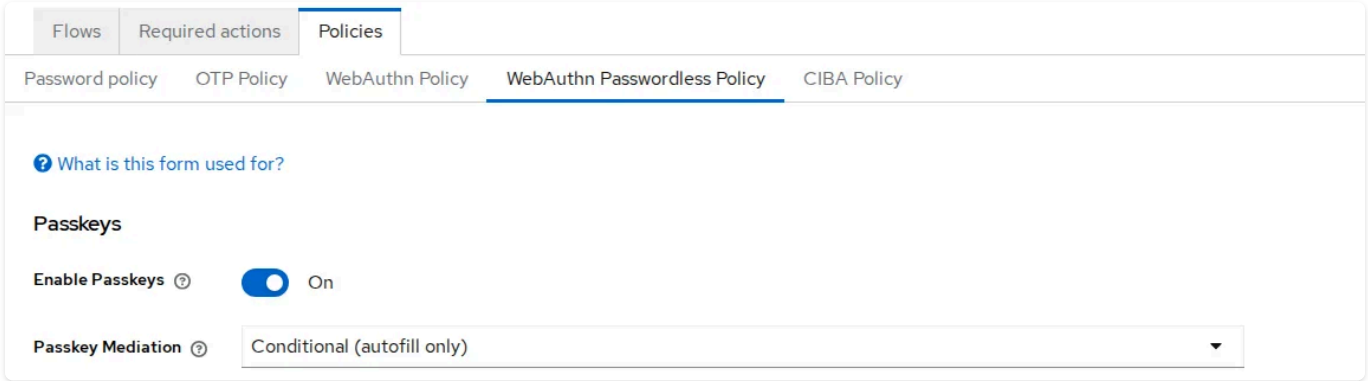
```
spring.security.oauth2.client:
  registration.keycloak:
    client-id: nerdshirts
    client-secret: ${KEYCLOAK_CLIENT_SECRET}
    scope: openid, profile, email
  provider.keycloak:
    issuer-uri: http://localhost:8180/realms/nerdshirts
    user-name-attribute: email
```

```
http.oauth2Login(oauth2 -> oauth2
        .userInfoEndpoint(u -> u.userAuthoritiesMapper(keycloakRoles())))
    .logout(logout -> logout.logoutSuccessHandler(keycloakLogout(clients)))
```

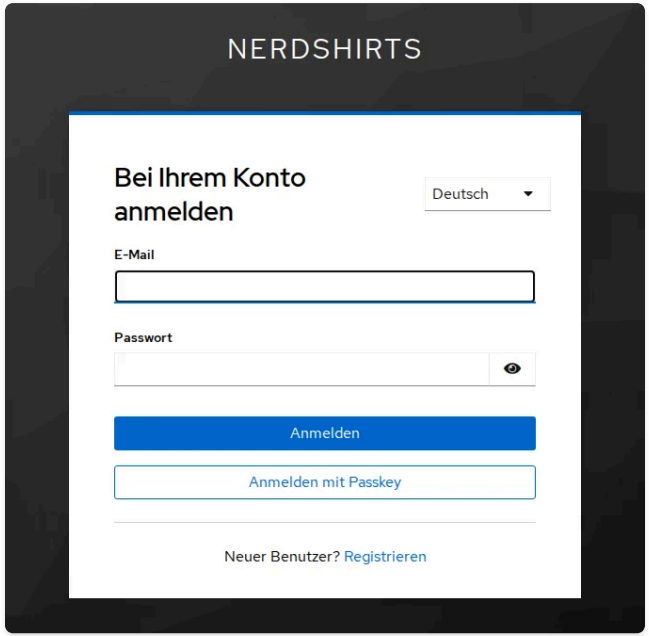
Java im Shop: **–474 / +65 Zeilen** – Login-Seite, Passkey-JS, Verwaltung, Nutzertabelle: weg

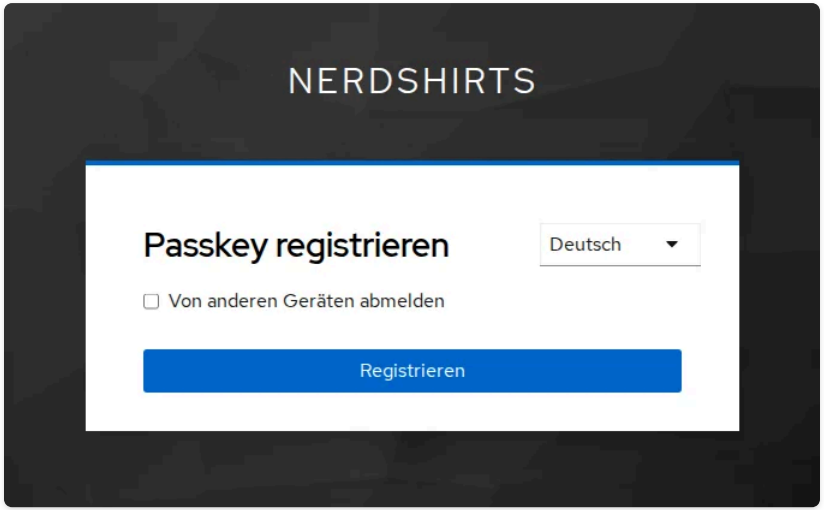
Passkeys in Keycloak einschalten

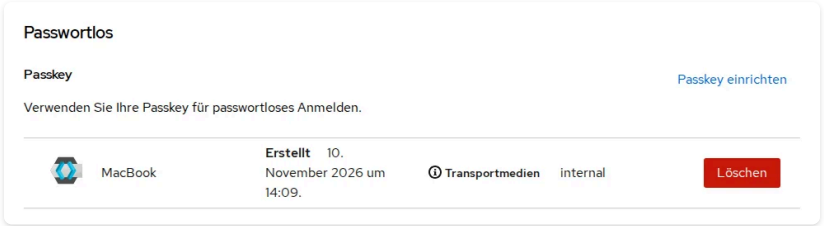
- **Authentication → Policies → WebAuthn Passwordless Policy**: „Enable Passkeys“ (auch unter **Realm settings → Login**)
- Gleiche Seite: RP-Name, User Verification `required`, Resident Key `required`
- **Required Action** „Webauthn Register Passwordless“ am Nutzer → Passkey beim nächsten Login



So sieht es für Nutzer aus







Live-Demo



Branch `step-6-keycloak`

⚖ Eigenbau, IdP oder Service?

	SPRING SECURITY	KEYCLOAK	CORBADO & CO.
Aufwand	mittel	mittel (Betrieb!)	gering
Kontrolle	voll	voll	eingeschränkt
Mehrere Apps	jede einzeln	zentral	zentral
Updates	Library	Keycloak-Releases	Anbieter
Abhängigkeit	Spring	Keycloak	Anbieter



Kapitel 5

Migration: von Passwörtern zu Passkeys

Niemand fängt auf der grünen Wiese an

- Bestehende Konten haben Passwort, vielleicht 2FA
- Nicht alle Nutzer haben ein passkey-fähiges Gerät (oder wollen eins)
- Eine harte Umstellung führt zu Support-Tickets und verlorenen Konten
- **→ Migration in Phasen**

Drei Phasen



1. Anbieten

Passwort ist Standard, Passkey optional und beworben





2. Bevorzugen

Passkey ist Standard beim Login, Passwort nur Fallback





3. Ablösen

Passkey ist der einzige Weg (+ Recovery), Passwort entfernt

Phase 1: Nach dem Login fragen

- Nutzer meldet sich mit Passwort an
- Danach: „Beim nächsten Mal ohne Passwort? Passkey anlegen“
- Ablehnen erlaubt, später erneut fragen (aber nicht bei jedem Login)
- Im Shop: Interstitial nach dem Login, in Keycloak: Required Action

☆ Phase 2: Passkey direkt im Login-Feld

```
<input name="username" autocomplete="username webauthn">
```

```
const credential = await navigator.credentials.get({
  mediation: "conditional",
  publicKey: options,   // von /webauthn/authenticate/options
});
```

- **Conditional UI:** Der Browser schlägt Passkeys im Autofill vor
- Wer keinen hat, tippt einfach wie gewohnt sein Passwort

🪄 Noch sanfter: automatisch upgraden

```
await navigator.credentials.create({
  mediation: "conditional",
  publicKey: registrationOptions,
});
```

- Nach einem Passwort-Login aus dem Passwortmanager legt der Browser **still** einen Passkey an
- Kein Dialog, keine Entscheidung für den Nutzer
- Nur wenn Browser und Passwortmanager das unterstützen → Feature-Detection

Recovery: die eigentliche Frage

- **Mehrere Passkeys pro Konto** (Kapitel 3) sind die beste Recovery
- Synchronisierte Passkeys überleben den Geräteverlust
- Fallback: E-Mail-Verifikation + neuen Passkey registrieren
- Für hohe Sicherheit: Recovery-Codes, Identifikation über den Support
- **Keine Sicherheitsfragen.**



Merksatz

Ein Konto ist nur so sicher wie sein schwächster Login-Weg.



Für den Server
ändert sich
nichts

Ausblick

- **iPhone** als Authenticator: QR-Code, Bluetooth, Key bleibt am Handy
- **Synchronisiert**: überlebt Geräteverlust, liegt beim Anbieter
- **YubiKey**: verlässt das Gerät nie – verloren = weg
- **Umziehen** zwischen Anbietern: FIDO Credential Exchange
- **Native Apps** teilen Passkeys mit der Website



Und jetzt:

Produktionsreif?

Was ich mitnehme

- Die Spring-Magie ist ein guter Start – die Arbeit steckt in Persistenz, UX und Lebenszyklus
- Die RP-ID ist eine Entscheidung für immer
- Mehrere Passkeys pro Konto sind die beste Recovery
- Migration geht sanft: anbieten, Autofill, automatisch upgraden
- Mehrere Apps? Dann gehören Passkeys in den IdP

Zum Weiterlesen

- Spring Security Reference: Passkeys
- Keycloak Server Administration Guide: Passkeys / WebAuthn
- passkeys.dev – Libraries, Patterns, Browser-Support
- Demo-Code: `github.com/ch4inl3ss/demo-webshop`
- INNOQ Security Podcast #028: „Passkeys. Nutzbar und sicher?“

Danke! Fragen?

INNOQ
www.innoq.com



Felix Schumacher
felix.schumacher@innoq.com

innoQ Deutschland GmbH

Krischerstr. 100
40789 Monheim am Rhein
(+49) 2173 3366 0

Ohlauer Str. 43
10999 Berlin

Ludwigstr. 180 E
63067 Offenbach

Kreuzstr. 16
80331 München

Wendenstraße 130
20537 Hamburg

Spichernstraße 44
50672 Köln

Links

- Demo-Code: github.com/ch4inl3ss/demo-webshop
- Browser-Support: caniuse.com/passkeys
- Teil 1 als Video: youtube.com/watch?v=CFILx3VUN5A