

Autorizacija



ОБАВЕШТЕЊЕ ЗА СТУДЕНТЕ

- Настава на предмету Развој безбедног софтвера подразумева изучавање различитих механизма којима се нарушава информациона безбедност и врше напади на интернет апликације и софтверске системе.
- Студенти на предмету Развој безбедног софтвера могу ове методе за потребе изучавања да користе искључиво у оквиру затвореног лабораторијског окружења које је обезбеђено за наставу на предмету Развој безбедног софтвера.
- Студенти не могу да подразумевају да су на било који начин охрабрени од стране наставника или да им се препоручује да користе ове методе који се изучавају према другим апликацијама Електротехничког факултета или апликацијама било ког трећег правног или физичког лица.
- Свака евентуална активност коју би предузео неки студент коришћењем ових метода и механизма према апликацијама које нису у оквиру лабораторије на предмету искључива је одговорност студента.

Uvod

- Autorizacija je sigurnosna usluga koja određuje koje podatke korisnik može da vidi i koje akcije korisnik može da izvrši.
 - Odgovor na pitanje: Šta korisnik može da uradi?
- Nasuprot autentifikaciji koja utvrđuje korisnikov identitet.
 - Odgovor na pitanje: Ko je korisnik?

Modeli autorizacije

- ACL: Access Control List
- RBAC: Role-Based Access Control
- ABAC: Attribute-Based Access Control
- RAdAC: Risk Adaptive-Based Access Control

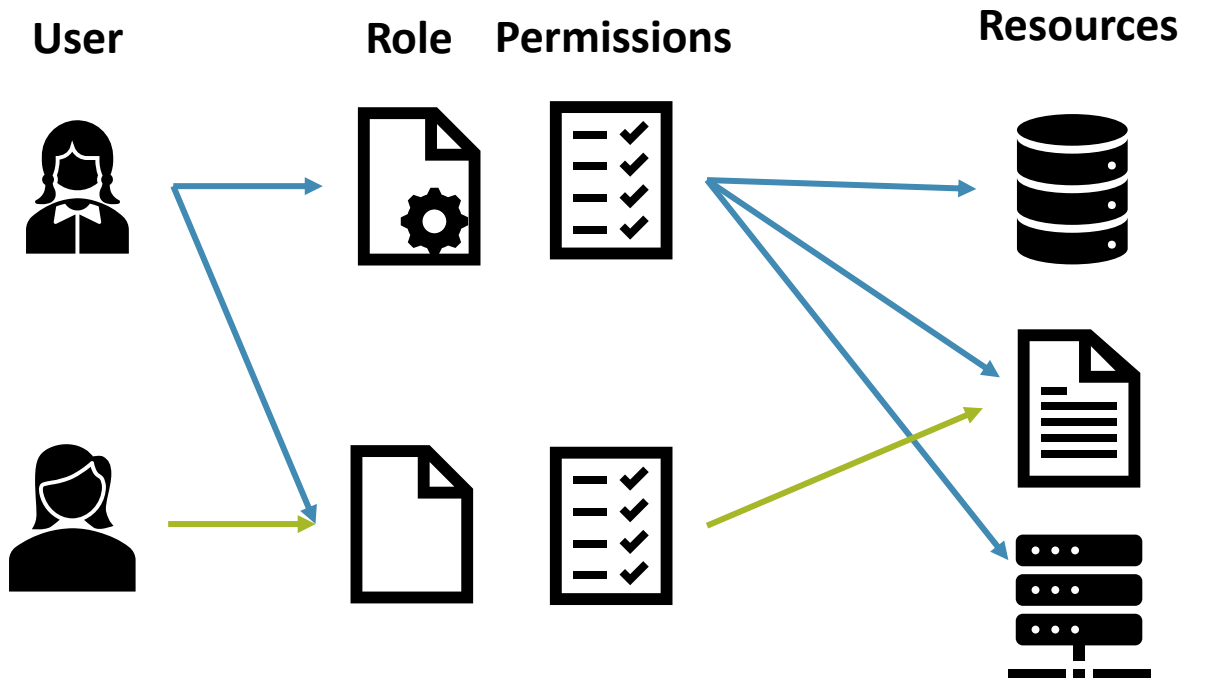
ACL: Access Control List

- Lista korisnika koji mogu da pristupe resursu
- Kada korisnik pokuša da pristupi resursu – lista pristupa daje odgovor: da ili ne.



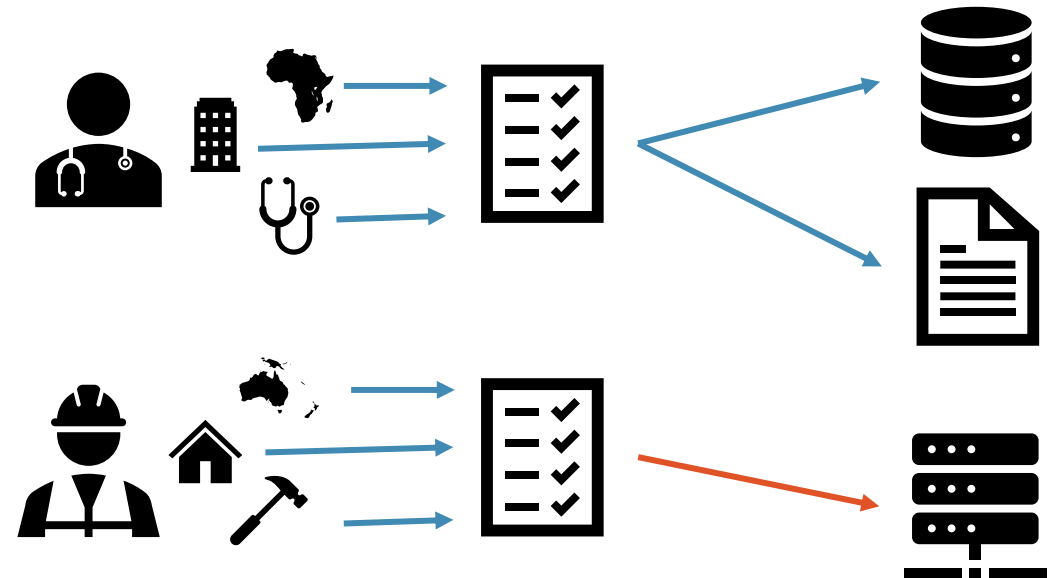
RBAC: Role-Based Access Control

- Korisnik ima jednu ili više rola
- Rola ima jednu ili više permisija za pristup resursu



ABAC: Attribute based access control

- Korisnik ima određene attribute (država, godine, smer na fakultetu, sektor u firmi...)
- Na osnovu kombinacije atributa korisnika se određuje da li ima pristup određenom resursu ili ne.



RAdAC: Risk Adaptive-Based Access Control

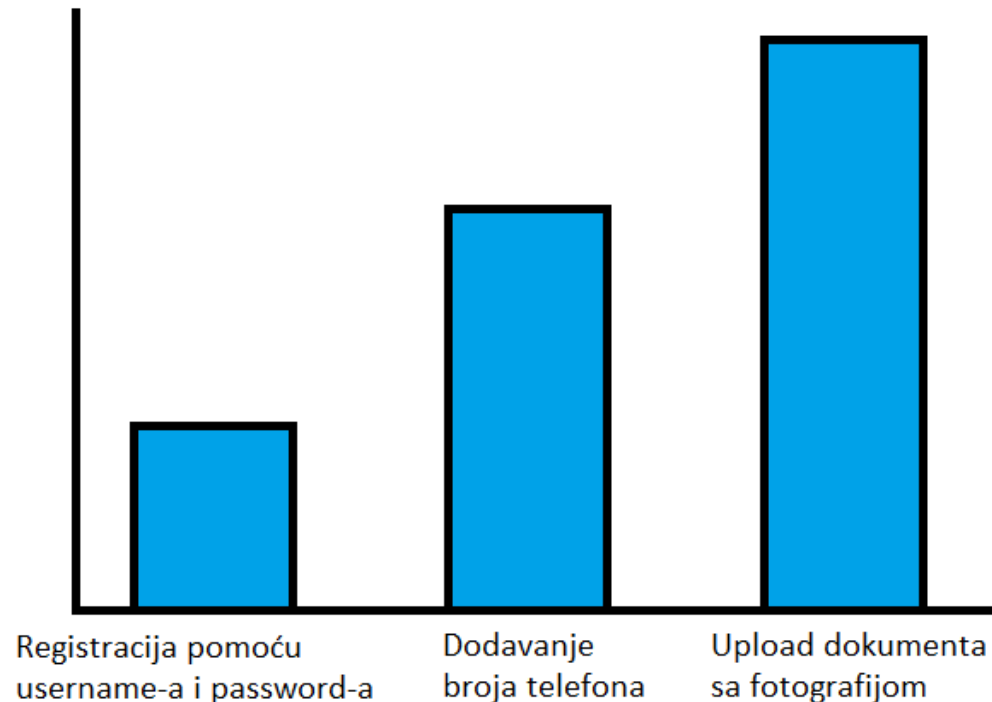
- Na osnovu analize korisnikovog ponašanja i trenutnih faktora određuje se nivo rizika.
 - Lokacija korisnika, uređaj sa koga se pristupa, broj prethodno pogrešnih unosa kredencijala...
- Svakom resursu je dodeljen granica rizika iznad/ispod koje se ne dozvoljava pristup.

RAdAC: Risk Adaptive-Based Access Control

- Primer sa realnog projekta:

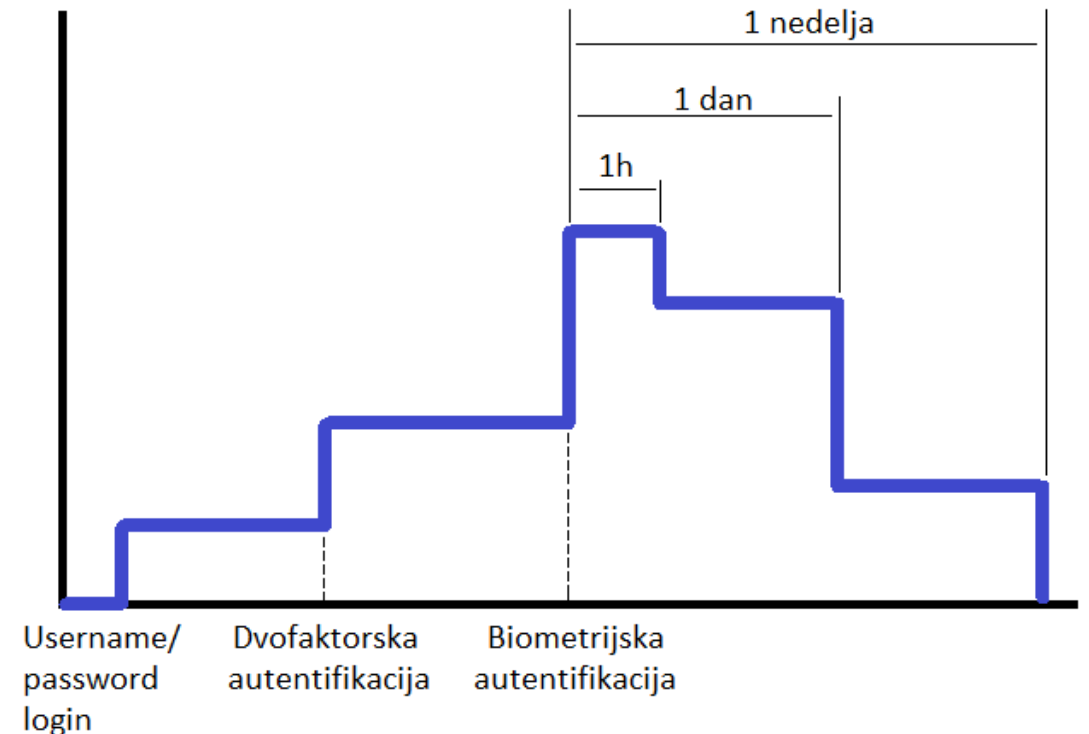
AccessLevel - važi trajno

(Koliko dobro sistem poznaje korisnika)



RegLevel - vremenom opada

(Koliko je validna poslednja autentifikacija)



Samostalni rad

Opis: Popunite matricu permisija po RBAC modelu koje bi po vašem mišljenju trebalo implementirati.

Vreme: **10 min**

Dodatni zadatak

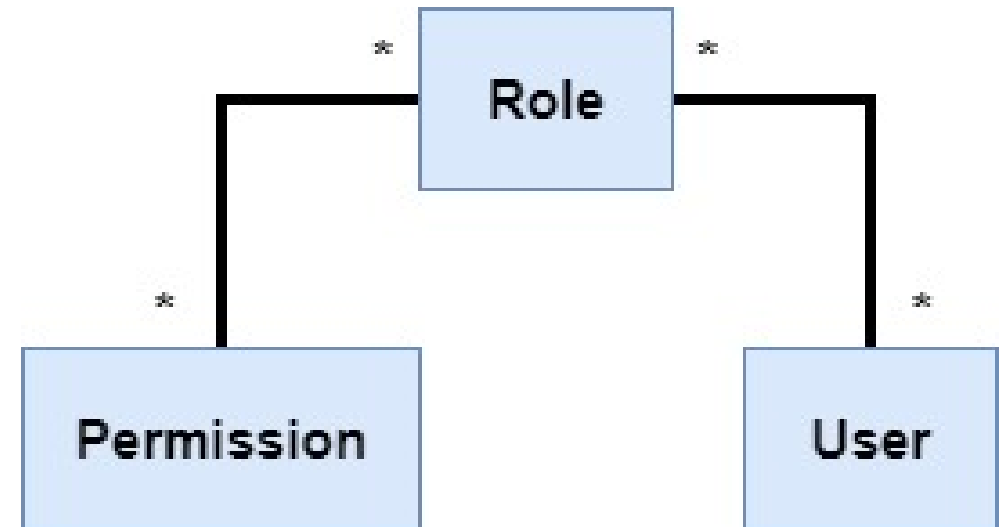
- Nedostaje jedan ceo use case u matrici, povezan sa stranicom korisničkog profila.
- 2 granularnije autorizacije na stranama promena detalja o automobilu i pregleda korisničkog profila
- Ukupno 3 dodatna reda

Samostalni rad

Use Case	Prodavac	Kupac
Lista automobila	x	x
Pretraga automobila		
Kupovina automobila		
Pregled i kreiranje komentara		
Pregled detalja automobila		
Promena detalja automobila		
Pregled bilo kog korisničkog profila		
Lista korisnika		
Pretraga korisnika		
Unapređenje korisnika u prodavca		
Pregled i promena svog korisničkog profila		

Kako implementirati autorizaciju?

1. Implementirati autorizacioni model u bazi podataka (tabele: korisnik, rola, permisija)
2. Implementirati učitavanje permisija za prijavljenog korisnika
3. Provera permisija pri pristupu resursima
4. Opciono: Poseban UI za administraciju korisnika, rola i permisija



Kako proveravati permisije?

- Da li je dovoljno samo prikazivati i sklanjati UI elemente?
 - Primer: Ukoliko ne želimo da dozvolimo da korisnik kupi kola, da li je dovoljno da samo sakrijemo dugme?

Kako proveravati permisije?

- Da li je dovoljno samo prikazivati ili sklanjati UI elemente?
 - Primer: Ukoliko ne želimo da dozvolimo da korisnik kupi kola, da li je dovoljno da samo sakrijemo dugme?
- Važno je proveravati permisije na frontendu i na backendu.
 - Na frontendu sakrivamo elemente samo zbog korisničkog iskustva (UX)
 - Proverom na serveru osiguravamo da zlonamerni korisnik ne može da pošalje direktno HTTP zahtev za neautorizovanu akciju

Primer

- Online članak u statusu „neobjavljeno“ (*draft*) kojem su mogli da pristupe direktno preko URLa ljudi koji nisu novinari

(npr. u slučaju da se poznata ličnost nađe u bolnici i da joj je život ugrožen, novinari prave *draft* članke o smrti te osobe, kako bi u slučaju da osoba zaista premine mogli da objave vest u roku od par minuta)

Kako proveravati permisije?

- Provera permisija je u principu samo IF statement

```
if(user.hasPermission("CAR_EDIT")) { ... }
```

- Šta bi bio bolji način za proveru permisija na serveru?

Kako proveravati permisije?

- Šta bi bio bolji način za proveru permisija na serveru?
 - Permisije su „*cross-cutting concern*“ – nisu deo jedne vertikale (jedna funkcionalnost sistema), nego su deo horizontale sistema – svuda su primenljive.
 - Želimo da primenjujemo permisije bez menjanja poslovne logike u metodama kada god je to moguće.
- Koji koncept iz programiranja znamo da najbolje rešava ovaj problem?

Kako proveravati permisije?

- Koji koncept iz programiranja znamo da najbolje rešava problem “*cross-cutting concern*”?
- Najbolje rešenje je neka vrsta *aspektno orijentisanog programiranja*.
 - anotacije u JAVI
 - atributi u .NET
- Nije svaka anotacija aspekt – framework definiše koje su anotacije aspekti

Spring Security

- I za role i za permisije koristi se klasa *GrantedAuthority*
- Razlika je jedino u tome što su imena rola sa prefiksom **ROLE_**
- Permisije se prilikom prijave korisnika učitavaju iz baze koristeći role korisnika
- Na osnovu permisija pročitanih iz baze kreiraju se odgovarajući **GrantedAuthorities** i dodaju u listu **GrantedAuthorities** prijavljenog korisnika
- Mi ćemo se fokusirati na permisije

Spring Security

- Koristi se **@PreAuthorize** anotacija za primenu permisije na resurs (endpoint)
- **CAR_DETAILS_EDIT** je naziv permisije

```
@PostMapping("/cars/{id}")  
@PreAuthorize("hasAuthority('CAR_DETAILS_EDIT')")  
public String editCar(@PathVariable("id") int id, Car car) {  
    carRepository.update(id, car);  
    return "redirect:/cars?id=" + id;  
}
```

Thymeleaf

Koristi se **sec:** XML namespace za primenu permisija HTML template

sec:authorize atribut koristi se za uslovni prikaz HTML elemenata

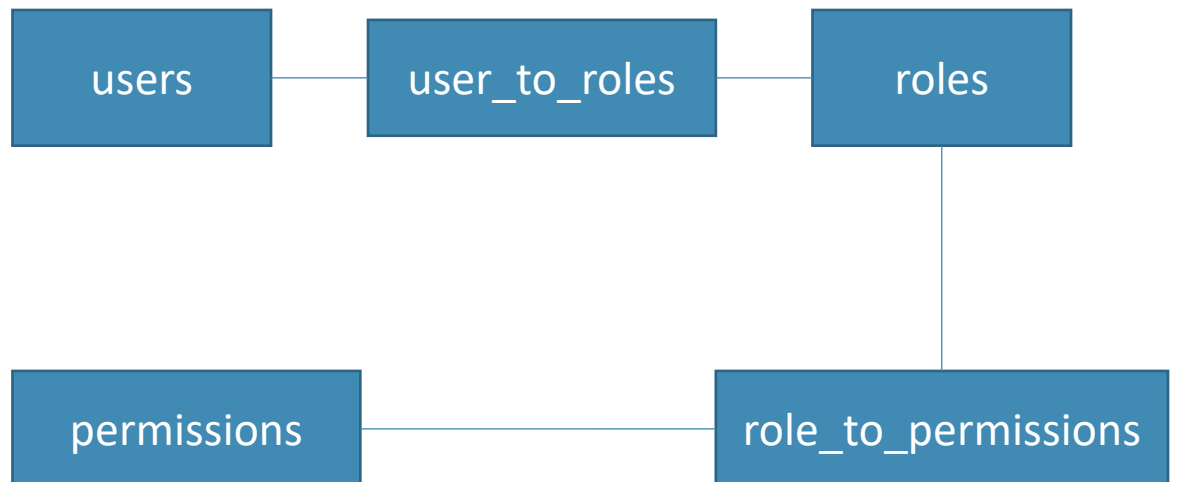
```
<div sec:authorize="hasAuthority('CAR_DETAILS_EDIT')">  
    <button type="submit" class="btn btn-primary">Save</button>  
</div>
```

Implementacija autorizacije

Demonstracija na promeni detalja automobila

Implementacija autorizacije

- Na slici su tabele koje se nalaze u bazi a relevantne su za autorizaciju
- Baza se popunjava podacima koristeći SQL skript u fajlu **data.sql**
- U **data.sql** već se nalaze:
 - Sve permisije (tabela **permission**)
 - Sve role (tabela **role**)
 - Korisnik **bruce** ima **SELLER** rolu
 - Korisnik **peter** ima **BUYER** rolu
 - Obe role **SELLER** i **BUYER** imaju **CAR_LIST_VIEW** permisiju



Implementacija autorizacije

- Za povezivanje entiteta, tabele **user_to_roles** i **role_to_permissions** koriste **id** vrednosti entiteta
- Ukoliko želimo da korisnik **bruce** ima rolu **SELLER**, a rola **SELLER** permisiju **CAR_LIST_VIEW** potrebni su sledeći podaci u bazi

users		user_to_roles		roles		role_to_permissions		permissions	
id	username	userId	roleId	id	name	roleId	permissionId	id	name
1	bruce	1	1	1	SELLER	1	1	1	CAR_LIST_VIEW

Implementacija autorizacije

- Često za administraciju rola i permisija postoji poseban korisnički interfejs
- U našoj aplikaciji to nije slučaj, pa ćemo menjati role i permisije direktno u bazi podataka, odnosno fajlu **data.sql**

Implementacija autorizacije

- Za učitavanje permisija koristi se klasa *PermissionService*
- Primer, učitavanje permisija za korisnika:

```
List<Permission> permissions = permissionService.get(user.getId());
```

- Kada se korisnik prijavi, poziva se metoda *DatabaseAuthenticationProvider.authenticate*
- U slučaju uspešne prijave, koristeći *PermissionService* učitavaju se permisije, odnosno *granted authorities*
- Korisnika i učitane permisije dodajemo u objekat *UsernamePasswordAuthenticationToken* čime Spring Framework postaje svestan permisija koje prijavljeni korisnik ima

Implementacija autorizacije

- Relevantni delovi *authenticate* metode *DatabaseAuthenticationProvider* klase

```
public Authentication authenticate(Authentication authentication) throws AuthenticationException {
    String username = authentication.getName();
    String password = authentication.getCredentials().toString();

    Object details = authentication.getDetails();
    Integer totp = StringUtils.isEmpty(details) ? null : Integer.valueOf(details.toString());

    boolean success = validCredentials(username, password);
    if (success) {
        User user = userRepository.findUser(username);
        List<GrantedAuthority> grantedAuthorities = getGrantedAuthorities(user);
        return new UsernamePasswordAuthenticationToken(user, password, grantedAuthorities);
    }

    throw new BadCredentialsException(String.format(PASSWORD_WRONG_MESSAGE, username, password));
}

private List<GrantedAuthority> getGrantedAuthorities(User user) {
    List<Permission> permissions = permissionService.get(user.getId());
    List<GrantedAuthority> grantedAuthorities = new ArrayList<>();
    for (Permission permission : permissions) {
        grantedAuthorities.add(new SimpleGrantedAuthority(permission.getName()));
    }
    return grantedAuthorities;
}
```

Implementacija autorizacije

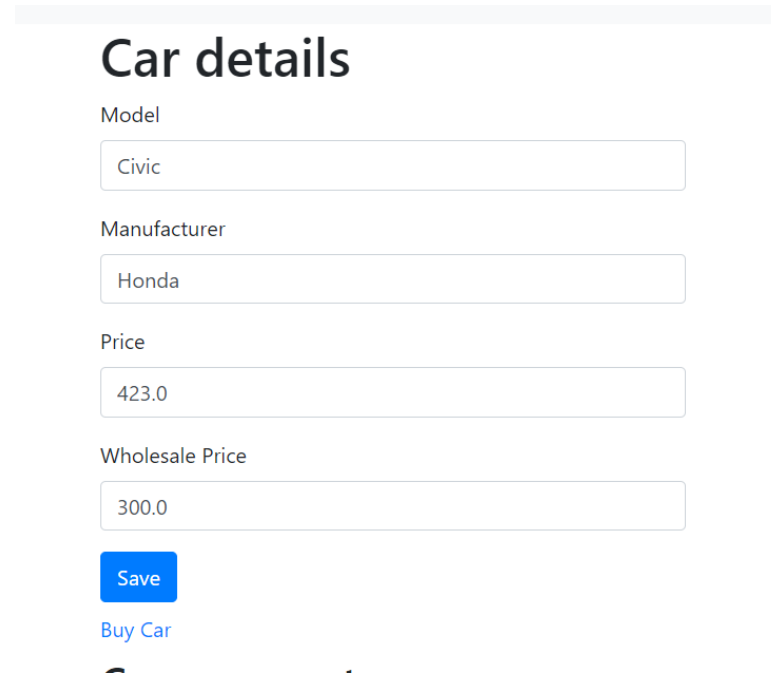
- Za promenu detalja automobila postoji permisija **CAR_DETAILS_EDIT**
- U fajlu **data.sql** vidimo da je **id** ove permisije **7**
- Želimo da rola **SELLER**, čiji je **id 1**, ima ovu permisiju
- U tabelu **role_to_permission** dodajemo sledeće

```
insert into role_to_permissions(roleId, permissionId)
values
(1, 1),
(2, 1),
(1, 7);
```

- Nakon ove izmene, korisnik **bruce** koji ima rolu **SELLER** imaće permisiju **CAR_DETAILS_EDIT**

Implementacija autorizacije

- Želimo da samo korisnici koji imaju permisiju **CAR_DETAILS_EDIT** mogu da menjaju polja kao što su **Model** ili **Price**, i da vide **Save** dugme na **Car details** stranici



The screenshot displays a web form titled "Car details". It contains four input fields: "Model" with the value "Civic", "Manufacturer" with the value "Honda", "Price" with the value "423.0", and "Wholesale Price" with the value "300.0". Below these fields is a blue "Save" button and a blue link labeled "Buy Car".

Car details

Model

Civic

Manufacturer

Honda

Price

423.0

Wholesale Price

300.0

Save

[Buy Car](#)

Implementacija autorizacije

- **Car details** stranici odgovara **car.html**
- Na formi za promenu detalja automobila potrebno je dodati sledeće:

```
<form class="col-5" method="POST" th:action="@{/cars/_${car.id}__}">
  <div class="form-group">
    <label for="model">Model</label>
    <input type="text" id="model" name="model" class="form-control" th:value="${car.model}"
      th:disabled="${not #authorization.expression('hasAuthority(''CAR_DETAILS_EDIT'')')}">
    </div>
    <div class="form-group">
    <label for="manufacturer">Manufacturer</label>
    <input type="text" id="manufacturer" name="manufacturer" class="form-control" th:value="${car.manufacturer}"
      th:disabled="${not #authorization.expression('hasAuthority(''CAR_DETAILS_EDIT'')')}">
    </div>
    <div class="form-group">
    <label for="price">Price</label>
    <input type="text" id="price" name="price" class="form-control" th:value="${car.price}"
      th:disabled="${not #authorization.expression('hasAuthority(''CAR_DETAILS_EDIT'')')}">
    </div>
    <div class="form-group">
    <label for="wholesalePrice">Wholesale Price</label>
    <input type="text" id="wholesalePrice" name="wholesalePrice" class="form-control" th:value="${car.wholesalePrice}"
      th:disabled="${not #authorization.expression('hasAuthority(''CAR_DETAILS_EDIT'')')}">
    </div>
    <div sec:authorize="hasAuthority('CAR_DETAILS_EDIT')">
      <button type="submit" class="btn btn-primary">Save</button>
    </div>
  </form>
```

Implementacija autorizacije

- Ukoliko se sada prijavimo kao korisnik **peter** koji nema **EDIT_CAR_DETAILS** permisiju, **Car details** stranica izgleda ovako

Car details

Model

Civic

Manufacturer

Honda

Price

423.0

Wholesale Price

300.0

[Buy Car](#)

Car comments

Implementacija autorizacije

- Nije dovoljno samo sakriti elemente na frontendu, već se provera mora vršiti i na backendu
- Metoda (endpoint) koja se izvršava prilikom promene detalja automobila je *editCar* u klasi *CarsController*
- Potrebno je dodati sledeće:

```
@PostMapping("/cars/{id}")  
@PreAuthorize("hasAuthority('CAR_DETAILS_EDIT')")  
public String editCar(@PathVariable("id") int id, Car car) throws SQLException {  
    carRepository.update(id, car);  
    return "redirect:/cars?id=" + id;  
}
```


Priprema za samostalni rad

Otvorite IntelliJ IDEA

```
git checkout master
```

```
git pull
```

Samostalni rad

Opis:

- Povezati neophodne permisije sa rolama (iz matrice u data.sql role_to_permissions)
- Koristeći Spring Security, implementirati proveru permisija (backend i frontend) za: **USER_PROFILE_VIEW**
- **USER_PROFILE_VIEW** je permisija koja dozvoljava da se vidi profil na stranici Persons
- Struktura već postoji u bazi i učitavanje podataka za permisije je već implementirano
- Bruce je SELLER / Peter je BUYER
- Vreme: **40 minuta**

Dodatni zadatak

- Koristeći Sprint Security, implementirati proveru permisija (backend i frontend) za: **CAR_WHOLESALE_PRICE_VIEW**
- **SecurityUtil.hasPermission** služi za proveru permisija u Java kodu

Rešenje zadatka

Demonstracija osnovnog zadatka

Da li je dovoljno samo sakriti stranicu?

Da li je dovoljno samo sakriti stranicu?

```
fetch('http://localhost:8080/persons/1', {method: 'DELETE'});
```

Zaključak:

- Iako je „Brisanje korisničkog naloga“ zasebna permisija (npr. USER_DELETE), važno je posmatrati celokupni autorizacioni sistem.
- Ukoliko korisnik ne može da vidi profil i Delete dugme, ne bi trebalo ni da može da pozove endpoint.

Da li je dovoljno samo sakriti stranicu?

- Potrebno je da autorizacioni sistem bude konzistentan i razmišlja o celokupnoj slici korisničkih zahteva.
- **Korisnik koji ne vidi stranicu sa nekom operacijom, ne bi smeo da ima mogućnost da izvrši tu operaciju slanjem HTTP zahteva.**

2021

A01:2021-Broken Access Control

A02:2021-Cryptographic Failures

A03:2021-Injection

A04:2021-Insecure Design

A05:2021-Security Misconfiguration

A06:2021-Vulnerable and Outdated Components

A07:2021-Identification and Authentication Failures

A08:2021-Software and Data Integrity Failures

A09:2021-Security Logging and Monitoring Failures*

A10:2021-Server-Side Request Forgery (SSRF)*

Rešenje zadatka

Demonstracija dodatnog zadatka