

**IT381 – Zastita i bezbednost informacija**

Projektna dokumentacija – Bezbednost baze podataka

**Profesor/Asistent:**

prof. Milena Bogdanović

Goran Stemenović

**Student:**

Tamara Mišić, 4104

**Predlog teme**

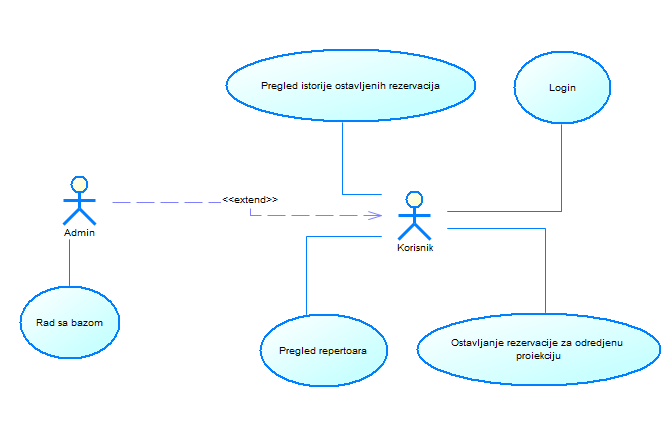
Baza podataka predstavlja bitan deo većine sistema koji se postavljaju na veb serveru ili su to ipak mobilne aplikacije. U svakom slučaju, najveći broj sistema nezavisno od navedenog tipa, koristi bazu podataka kao sredstvo za skladištenje bitnih podataka koji su potrebni da se sačuvaju ili kasnije u nekom trenutku pročitaju. Ideja je da na običnom sistemu kao što je izabran, tj. veb sajt za bioskop, pokazati kako se može uništiti bezbednost podataka u bazi podataka I kako se ti podaci mogu zloupotrebiti, kao i na koji način se oni štite i čine bezbednim.

**Uvod u projekat**

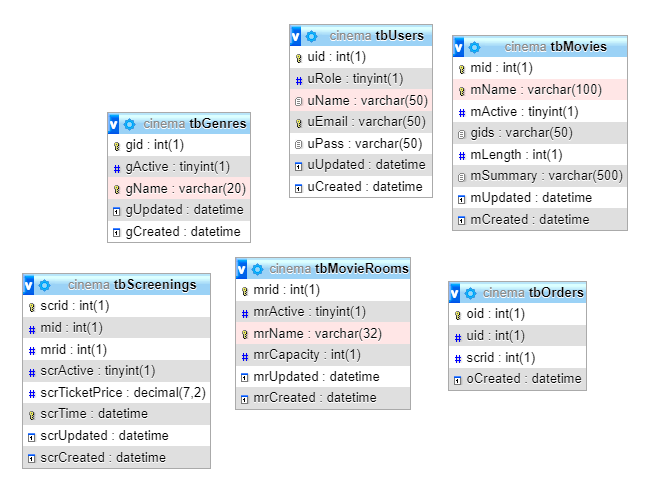
Glavni cilj projektnog zadatka je pre svega prikaz repertoara bioskopa i rezervacija karata za filmove koje on nudi. Konekcija sa bazom uspostavljena je pomoću php funkcije mysqli\_connect koja se nalazi u posebnom fajlu koji se dalje uključuje na svakoj strani pomoću php funkcije require\_once. Podaci su preuzeti iz baze preko sql upita i funkcije my\_sqli\_query.

Glavna funkcionalnost je naravno pregled repertoara I rezervisanja karata. To se postiže na sledeći način: Korisnik ispod slider-a sa slikama filmova koji se daju trenutnog meseca u bioskopu može da primeti dropdown input sa datumima za koje bioskop nudi repertoar. Kada izabere određeni datum uz pomoć ajax-a on može bez submit-ovanja forme da dobije repertoar za odabrani datum. Tu može da vidi sve potrebne detalje za projekcije koje se daju odabranog datuma: u koliko sati se daje projekcija, sliku filma, naziv filma, cenu karte, u kojoj sali se daje, žanrove I kratak opis radnje filma. Za svaku projekciju postoji dugme preko kog on može da rezerviše kartu za određeni film. Ukoliko nije ulogovan, odmah ga preusmerava na stranicu sa logovanjem ili ako nema nalog sa registrovanjem. Validacija svih formi odrađena je pomoću jquerz ugrađene metode validate(). Nakon logovanja, odmah ga preusmerava na stranu za rezervaciju sa jedinstvenim generisanim QR kodom za njegovu kartu. U sesiji se pamti za koju je projekciju korisnik hteo da rezeviše kartu kako ne bi morao opet da se vraća na početnu stranu i da pronalazi film , već to odmah uradi i dobije QR kod. Kada je korisnik ulogovan on ima pristup stranici My account na kojoj vidi svoju istoriju odgledanih filmova, gde može da vidi više detalja o njima ili da ih obriše. Admin sa druge strane, nakon pristupa backend-u i uspešnog logovanja jer je to prva strana backend-a kako bi se zaštitio od pristupa korisnika, on dobija uvid svih podataka iz baze preko tabela, gde može da vidi više detalja o tim podacima, da ih menja(update), da dodaje nove ili da ih briše(tj. da ih deaktivira; ovo je urađeno na ovaj način kako bi korisniku i dalje mogla na stranici My account da se prikaže istorija odgledanih filmova) . Admin i korisnik razlikuju se po uRole polju u bazi koji može da bude 0(korisnik) ili 1(admin). Admin nakon logovanja može da pristupi i frontend-u kao logovan korisnik, dok obrnuto ne može, tj. da korisnik pristupi backend-u.

*Use-Case dijagram:*



*Model baze podataka:*

  
**Uvod u tehnologije**

Za frontend korišćen je bootstrap framework, jQuery i AJAX, dok je za backend korišćen PHP.

Jquery je korišćen u svrhu validiraja forme prilikom rezervacije mesta za projekciju filma, pomoću metode validate(), dok je AJAX korišćen da bi u real-time prilikom izbora dana iz padajuće liste korisnik odmah dobio repertoar bioskopa za taj izabran dan. Čitav rad sa bazom funkcionise pomoću PHP-a.

Najpre struktura smeštanja svih podataka i koda za sajt je sledeća: u folderu projekta se nalaze svi fajlovi koji su namenjeni za frontend prikaz, tu se nalaze i folderi sa slikama i css kodom. Ceo kod za backend podeljen je u poseban folder backend koji je na taj način odvojen od frontend-a

**SQL Injection**

SQL Injection(SQLi) predstavlja ranjivost veb bezbednosti koja omogućava nekome napadaču da modifikuje upite koji idu sa aplikacije do baze podataka. Uglavnom omogućava napadaču da ima pristup podacima za koje ne bi trebalo da ima uvid. U mnogim slučajevima napadač može zbog nedozvoljenog pristupa tim podacima i da ih modifikuje ili da ih obriše, a može i da sprovede DoS napad.

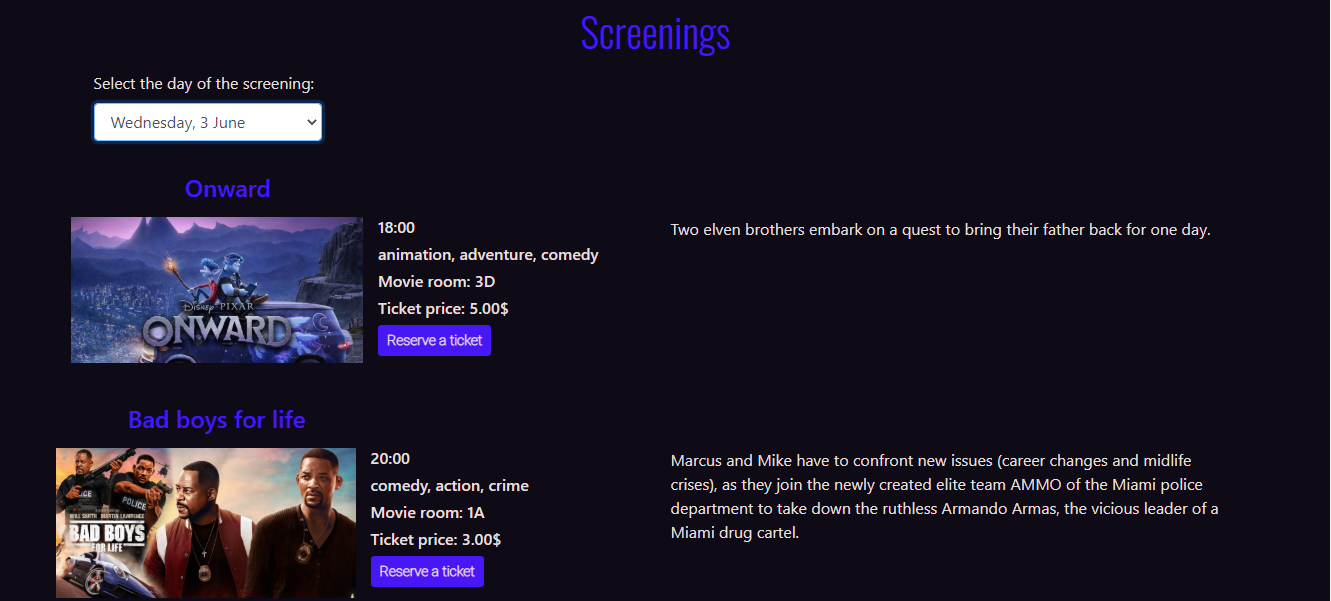
Rezultat ovakvog napada je neautorizovan pristup osetljivim podacima kao što su šifre, podaci o kreditnim karticama i ostali lični podaci. Postoji više vrsta SQLi napada:

* Pristup skrivenim podacima – modifikovanje upita kako bi se dobili dodatni podaci
* Promena aplikativne logike – modifikacija upita tako da promeni logiku aplikacije
* UNION napadi – modifikovanje upita tako da se dobiju podaci iz različitih tabela iz baze podataka
* Proučavanje baze podataka – modifikovanje upita tako da se dobiju informacije o strukturi i verziji baze podataka
* Slep SQL Injection – rezultati upita nisu vraćeni preko upita koji napadač kontroliše

Sve vrste ovih napada se mogu izvršiti preko unosa u formu, i zbog toga je bitno da se ona pravilno validira i iskoriste različite funkcije koje će sprečiti napade. U zavisnosti od tehologije u kojoj se radi postoje različite vrste metoda i načina sprečavanja, ali se sve uglavnom svode na isto, a to je uklanjanje specijalnih karaktera i provera tipa podataka koji se unosi, njegovog oblika i dužine.

Na primeru jednostavne aplikacije za bioskop prikazaće se te funkcionsanje I način upotrebe takvih metoda.

*Prikaz kako izgleda repertoar bioskopa:*



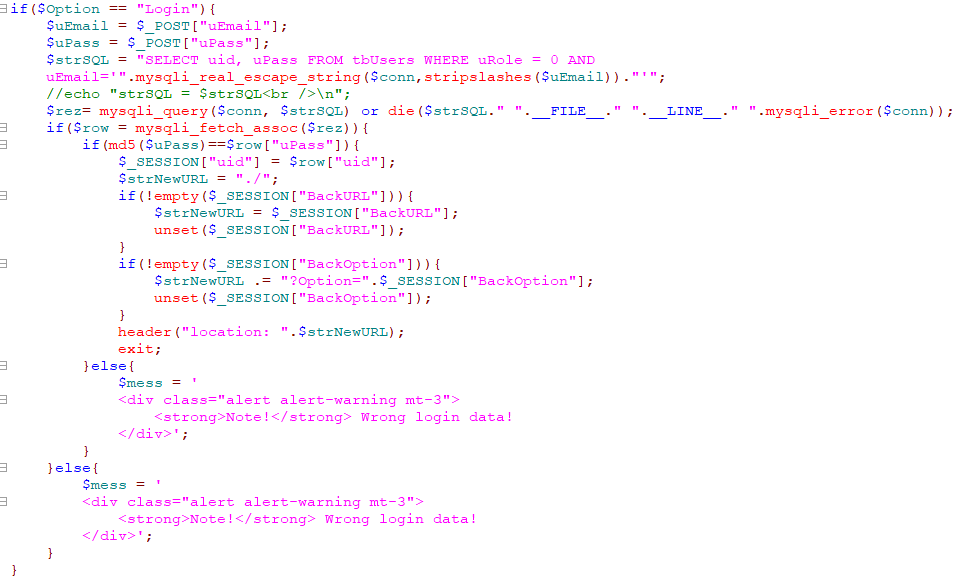
Ukoliko korisnik nakon pregledanog repetroara bioskopa želi za određenu projekciju da rezerviše karte klikom na dugme reserve a ticket, on mora prvenstveno da se uloguje. Forma za login je ranjiv deo sistema za bazu podataka jer se upravo popunjavanjem I manipulacijom iste može oštetiti baza podataka.

Kako izgleda kod za validaciju forme sa frontend strane:



Ovde vidimo da pomoću metode validate() proveravamo podatke koji su uneti u formu ali sa frontend strane, što znači da poslati podaci na backend strani su već provereni za osnovne kriterijume kao što su: prazan string, odgovarajući tip podataka, validna dužina podataka itd.

Provera na backend strani:



Problem sa SQL injection rešen je uz pomoć dve php funkcije: my\_sqli\_real\_escape\_string i stripslashes.

Prva funkcija sklanja sve specijalne karaktere kao što su (ASCII 0), \n, \r, \, ', ", Control-Z.

Druga funkcija sklanja sve unete slash-ove.

Postoji još drugih načina kako sprečiti SQL Injection, naprimer:

String query = "SELECT \* FROM products WHERE category = '"+ input + "'";

Statement statement = connection.createStatement();

ResultSet resultSet = statement.executeQuery(query);

U ovom slučaju upit se direktno konkatenira u upit, što nije dobra praksa I predstavlja veliku mogućnost za lako izvršavanje SQLi napada. Na lak način se ovaj kod može prepraviti tako da ne bude više ranjiv, a to je korišćenje parametrizovanih upita:

PreparedStatement statement = connection.prepareStatement("SELECT \* FROM products WHERE category = ?");

statement.setString(1, input);

ResultSet resultSet = statement.executeQuery();