Object Oriented Programming in PHP

1. Introducere

1. Tipuri de programare

- Abordarea declarativă a lucrului într-un limbaj de programare ascunde logica necesară pentru efectuarea unei acţiuni, unde, în schimb, este folosită o indicare a rezultatului dorit al unui cod către limbajul de programare – limbajul SQL este un bun exemplu în acest sens.
- Abordarea imperativă a lucrului într-un limbaj de programare implică faptul că folosim expresii care schimbă starea programului, codul, aplicaţia (if-else, de exemplu).
- Atât programarea procedurală, cât şi cea orientată pe obiecte aparţine paradigmei imperative.
- Principalul aspect al abordării procedurale a programării este funcţia care ne ajută să grupăm codul, să nu repetăm acelaşi cod şi să-l facem mai dinamic.
- În abordarea orientată pe obiecte, aplicaţia completă este împărţită în părţi mai mici, pe care le numim obiecte, care reprezintă o unitate de program care depinde, în principal, de ceea ce este destinat în aplicaţie.
- Abordarea de sus în jos (top-down) a programării poate fi găsită în abordarea procedurală, unde, de fapt, am stabilit mai întâi o problemă mare pe care trebuie să o rezolvăm, apoi o descompunem în unităţi mai mici pentru a o rezolva efectiv.
- Abordarea de jos în sus (bottom-up) a programării poate fi găsită în programarea orientată pe obiecte, unde scriem mai întâi clase şi metode pentru viitorul obiect, apoi le folosim şi le apelăm după necesitate.
- Modificatorii de acces, care controlează nivelul de acces în clase sau obiecte, ne permit să creăm câmpuri şi metode care nu pot fi accesate de nimeni din exterior.
- În programarea procedurală, funcţiile ocupă locul principal în cod şi sunt mai importante decât datele – putem spune că funcţionalitatea este mai importantă decât datele, iar în acest fel avem modelarea aplicaţiei bazată pe lumea ireală.
- În programarea orientată pe obiecte, avem modelarea bazată pe lumea reală, unde obiectul reprezintă baza pentru modelarea aplicaţiei complete.

2. Conceptele programării orientate pe obiecte

- Clasele reprezintă posibilitatea de creare a unui anumit şablon, template, pe baza căruia vor fi create ulterior unul sau mai multe obiecte.
- Obiectele reprezintă baza pentru programarea orientată pe obiecte. După ce am creat un şablon, adică o clasă, putem începe să instanţiem sau să creăm un obiect.
- Clasele pot seta anumite placeholdere, cu care funcţionează metodele din clasă. Acele placeholdere reprezintă, de fapt, câmpuri sau proprietăţi ale clasei.
- În practică, metodele clasei ar trebui observate ca funcţii - cu ele grupăm codul pentru o anumită funcţionalitate într-un singur loc, evităm repetarea şi îl putem face dinamic prin parametrii de intrare şi de ieşire.
- Câmpurile clasei şi metodele clasei sunt numite instanţiate, tocmai pentru că sunt definite în clasă, iar apoi sunt folosite doar atunci când **instanţiem un obiect** din clasă.
- Metodele şi câmpurile marcate cu cuvântul-cheie **static** se numesc membrii statici ai clasei şi nu necesită instanţierea obiectului şi pot fi apelate direct prin specificarea numelui clasei, apoi de două ori se introduc două puncte şi apoi numele câmpului sau al metodei.
- Încapsularea este un concept în care punem toate datele şi funcţionalitatea lor (respectiv câmpurile şi metodele) sub un singur acoperiş, adică un wrapper.
- Conceptul de moştenire ne permite să folosim codul unei alte clase fără a fi nevoie să-l duplicăm (putem folosi toate câmpurile şi metodele clasei părinte din clasa moştenitoare).
- Clasele abstracte reprezintă un şablon pentru crearea şabloanelor (claselor).
- Spunem că interfeţele reprezintă un concept care ne permite să definim ce metode trebuie să implementeze o clasă.
- Polimorfismul reprezintă posibilitatea de a avea o metodă definită care va face un lucru concret, dar implementarea

acelui lucru concret este lăsată în seama unei clase care moşteneşte dintr-o clasă abstractă sau implementează o interfaţă.

- Trait-urile (trăsăturile) ne permit să-i asigurăm unei clase să moştenească, adică să folosească mai multe trait-uri (similar cu utilizarea mai multor interfeţe) fără nicio restricţie.
- Este suficient să ne imaginăm spaţiul de nume ca pe o categorizare a diferitor clase, unde spunem că fiecare clasă aparţine unei anumite categorii (părţi a codului) sau că am plasat fiecare dintre clase în anumite casete care diferă prin ceea ce sunt destinate.
- Adnotările sunt metadate simple care pot fi incluse în orice parte a unei aplicaţii PHP, inclusiv în clase, funcţii, câmpuri şi metode.
- Reflecţia este capacitatea unui obiect de a face o examinare retrospectivă a lui însuşi şi de a ne informa despre metodele şi câmpurile sale în timpul execuţiei programului.
- Ceea ce ne permit şabloanele de design nu este să inventăm apă caldă, ci, atunci când întâlnim o problemă care este standard şi comună, să putem folosi un şablon anume pentru a rezolva o astfel de problemă.

2. Programarea orientată pe obiecte

3. Clasele şi obiectele

- Clasele şi obiectele sunt baza pentru programarea orientată pe obiecte.
- Clasa reprezintă un plan cu placeholdere (câmpuri) şi funcţionalităţi asupra acelor placeholdere (metode) din care vom instanţia obiecte.
- Clasele sunt, în general, separate prin fişiere, pentru o manipulare mai uşoară şi un control al încărcării ulterior şi sunt plasate într-un folder separat pentru clase.
- Creăm o clasă folosind cuvântul-cheie **class**.
- Instanţiem obiectul folosind cuvântul-cheie **new** şi specificând numele clasei.
- **Constructorul** ne ajută să inserăm imediat date în punctul în care instanţiem clasa în obiect şi astfel facilităm operarea şi claritatea codului.
- Destructorul (destructor) este o metodă care este întotdeauna apelată la distrugerea unui obiect, adică la ştergerea acestuia din memorie.
- Pentru a crea un obiect identic, dar separat într-un obiect, trebuie să-l clonăm cu cuvântul-cheie **clone**.

4. Membrii clasei, câmpurile şi metodele

- Modificatorii de acces ne permit să oferim acces controlat la câmpuri şi metode.
- În limbajul PHP avem modificatori de acces de tip privat, protejat şi public (private, protected şi public).
- Putem marca anumite câmpuri sau metode astfel încât să fie doar pentru uz intern (privat sau protejat/**private** sau **protected**, în funcţie de nivelul de utilizare), în timp ce putem marca anumite câmpuri sau metode astfel încât să fie pentru uz public (**public** – se pot apela şi utiliza atât în afara obiectului, cât şi în cadrul lui).
- Metoda magică numită **getter** ne permite să accesăm date private sau protejate şi să obţinem valoarea lor într-un mod formatat, aşa cum ne dorim. O marcăm cu **__get()**.
- **Setter** este o metodă magică care ne oferă posibilitatea de accesare a anumitor date şi de a le stabili valoarea într-un mod predefinit.
- Metoda magică __serialize ne serveşte la definirea comportamentului obiectului la care ne aşteptăm atunci când încercăm să-l serializăm, adică să-i salvăm starea la un moment dat.
- Metoda magică __unserialize() ne oferă spaţiu pentru a defini comportamentul în care vom dori să returnăm obiectul dintr-o formă serializată la forma clasică a obiectului.
- Metoda magică __toString() ne permite să definim ce se va întâmpla atunci când un obiect este apelat ca string.
- Metoda magică __isset() este, de fapt, apelată atunci când cineva încearcă să apeleze funcţia isset() sau empty() pe o proprietate inaccesibilă (privată sau protejată) sau inexistentă a obiectului.
- Metoda magică __unset() este apelată atunci când cineva încearcă să apeleze funcţia unset() pe o proprietate inaccesibilă (privată sau protejată) sau inexistentă a unui obiect.

5. Membrii statici şi de instanţă ai claselor

- Câmpurile şi metodele se definesc în clasă, dar le putem folosi doar atunci când instanţiem obiectul. De aceea, aceşti membri se numesc membri de instanţă ai clasei.
- Constantele nu sunt elemente dinamice precum câmpurile, aşadar, pentru ele nu trebuie să instanţiaţi obiectul. Acestea sunt elemente statice, adică membrii statici ai clasei şi, ca atare, se apelează prin clasa propriu-zisă.
- Cuvântul-cheie pentru definirea constantelor într-o clasă este **const**.
- Constantele se apelează prin specificarea numelui clasei, urmat de două puncte şi apoi de numele constantei.

- În clasă, pe lângă constante, putem defini şi câmpuri statice, pe care le marcăm prin cuvântul-cheie **static**.
- Câmpurile statice, deşi sunt statice, le putem modifica în timpul execuţiei codului, deoarece prin natura lor sunt câmpuri, adică variabile.
- Scopul lor este practic atunci când avem nevoie de o anumită valoare din clasa cu care vrem să lucrăm şi nu vrem să instanţiem un obiect complet cu toate celelalte câmpuri şi metode.
- Metodele statice ne permit să avem anumite metode de care nu avem nevoie pentru a crea un obiect de apelare.
- Metodele statice pot primi parametri, au parametri de ieşire, adică practic fac aproape tot ceea ce pot şi metodele de instanţă, doar metoda de apelare a acestora este diferită.
- Ele pot servi la optimizarea aplicaţiei, salvarea valorilor statice, partajarea aceloraşi valori între toţi membrii de instanţă ai clasei. Ele pot fi, de asemenea, folosite în clasele wrapper în loc de funcţiile obişnuite, asta pentru încărcarea mai uşoară a codului.

6. Încapsularea, moştenirea şi polimorfismul

- Încapsularea reprezintă un concept în care toate datele, variabilele şi funcționalităţile sunt conţinute într-un wrapper, respectiv într-o clasă, când vorbim despre un şablon pentru crearea obiectelor.
- Acest concept ne ajută să ştim în orice moment ce metode şi câmpuri aparţin unui nucleu, precum şi datele conţinute în acestea.
- Încapsularea ne ajută să definim, pentru anumite câmpuri, modul exact în care datele vor fi stocate şi furnizate, în ce format, fără a fi nevoie ca utilizatorul acelor câmpuri din obiect să ştie ceva despre el.
- Conceptul de moştenire se bazează pe relaţia dintre părinte şi copil.
- Acolo avem o clasă definită anterior, care are propriile câmpuri şi metode și este clasa părinte, şi o altă clasă care, de fapt, doreşte să folosească acele câmpuri şi metode dintr-o altă clasă şi să adauge unele dintre propriile câmpuri şi metode specifice, aceasta fiind clasa copil.
- În acest fel, este mult mai uşor să întreţinem codul, iar câmpurile şi metodele sunt definite o singură dată, într-un singur loc şi sunt folosite în mai multe locuri diferite fără probleme.
- În limbajul de programare PHP, o clasă poate moşteni o singură clasă.
- Moştenirea nu trebuie să se încheie doar la un singur nivel.
- Spunem că polimorfismul reprezintă un concept, al cărui scop constă în folosirea mai multor forme pentru a obţine aceeaşi esenţă.
- Ideea polimorfismului este să definim esenţa într-un singur loc (în interfaţă definim o metodă pe care o vor implementa clasele).
- Folosim acea esenţă în mai multe locuri, printr-o implementare specifică (clasele multiple implementează aceeaşi interfaţă).
- Implementarea este bună atât timp cât satisface esenţa, nu contează cum se realizează în locul în care este implementată, atât timp cât dă un rezultat bun.
- Polimorfismul ne ajută să nu repetăm codul, să rezolvăm probleme similare într-o formă diferită, dar cu aceeaşi esenţă etc.

7. Clasele abstracte

- Clasele abstracte se pot prezenta ca şabloane care vor fi utile pentru crearea ulterioară a altor şabloane.
- O clasă abstractă se află la un nivel peste o clasă regulată şi poate fi începutul unei clase sau a mai multor clase care moştenesc din aceasta (clasa abstractă).
- Cel mai practic scop al unei clase abstracte este acela de a defini în ea toate metodele abstracte cu posibilii parametri de intrare şi de a pregăti astfel crearea unei clase regulate.
- Când o clasă regulată moşteneşte o clasă abstractă, de fapt, am obligat clasa regulată să implementeze toate metodele găsite în clasa abstractă.
- Folosind clasele abstracte, putem vedea şi conceptul de polimorfism cu care ne-am familiarizat deja, în care o clasă

abstractă poate fi moştenită de mai multe clase regulate diferite.

8. Interfețe și trait-uri

- Interfețele ne permit să definim ce metode trebuie să implementeze o clasă care utilizează acea interfață.
- O clasă poate moșteni doar o singură clasă abstractă, în timp ce, în același timp, o clasă poate folosi mai multe interfețe simultan.
- Interfețele sunt create cu cuvântul-cheie **interface**, unde, după acel cuvânt-cheie, specificăm de fapt doar numele interfeței și parantezele acoladă în care vom enumera metodele pe care clasa va trebui să le implementeze mai târziu.
- Metodele pot specifica imediat ce parametri de intrare vor fi necesari în timpul implementării.
- Interfețele nu pot avea proprietăți, respectiv câmpuri, toate metodele trebuie să fie publice (public) și sunt automat abstracte și nu trebuie să aibă o implementare.
- Trait-urile rezolvă problema în care o clasă poate moșteni doar o clasă în PHP.
- O clasă poate folosi mai multe trait-uri și acestea pot avea metode definite și implementate, cât și proprietăți, câmpuri, împreună cu proprietățile și metodele statice ale acelor metode abstracte.
- Vedem că trait-ul în sine este creat folosind cuvântul-cheie **trait,** apoi specificând numele trait și folosindu-l în clasă folosind cuvântul-cheie **use.**

9. Adnotări și spații de nume

- Adnotările reprezintă metadate, respectiv informații despre o anumită parte a codului pentru înțelegerea lui și utilizarea mai ușoară.
- Adnotările pot fi incluse oriunde în cod, inclusiv între clase, metode, funcții și/sau câmpurile clasei.
- Adnotările sunt scrise cu semnul inițial „@” și specificând diferite date după aceea, în funcție de ce vrem să facem și de ce tip de adnotare folosim.
- Există adnotări încorporate cu care de obicei împiedicăm eliminarea erorilor din cod (ceea ce nu este cel mai bine, dar uneori necesar) și adnotări pe care le scriem în comentarii pentru a explica în detaliu și mai bine modul structurat al codului.
- Sintaxa PHPDoc este un exemplu bun de creare a documentației în cod folosind adnotări într-un mod foarte simplu.
- Spațiile de nume rezolvă problema utilizării mai multor clase cu același nume și gruparea claselor pe categorii specifice sau entități de aplicație.
- Definirea namespace-ului în jurul unei clase se face prin specificarea cuvântului-cheie **namespace**, punem un spațiu și apoi scriem imediat numelui namespace-ului dorit.
- Când dorim să folosim o clasă care are un namespace, înainte de instanțiere specificăm ce spațiu de nume apelăm, backslash, și apoi numele clasei.

10. Reflecţia

- Reflecția reprezintă posibilitatea unui obiect de a se privi singur în interiorul său și după aceea să ne ofere informații despre metodele și câmpurile lui în momentul execuției.
- Una dintre cele mai frecvente modalități de a folosi reflecția este în timpul debugging-ului/depanării codului PHP.
- get_class() returnează stringul cu numele clasei din care a fost creat obiectul.
- get_class_methods() returnează un șir cu toate metodele disponibile obiectului pentru utilizare la un moment dat.
- method_exists() ne ajută să verificăm dacă o metodă există într-un anumit obiect și, pe baza rezultatelor cercetării ei, să ramificăm în continuare codul.

11. Șabloane de proiectare (design patterns)

- Șabloanele sunt soluții standard la problemele de programare standard.
- **Singleton** permite crearea unei singure instanțe a unei anumite clase.
- **Observer** presupune „abonamentul” anumitor obiecte la informațiile emise de un alt obiect.
- Folosim **Factory** Pattern atunci când dorim să avem un intermediar în instanțierea claselor.
- **Decorator** este un înveliș în jurul clasei, în care adăugăm funcționalitate clasei înfășurate.
- **MVC** împarte sistemul (aplicația) în trei unități logice. Model (managerul de date), View (managerul de afișare a datelor) și Controller (managerul logicii de afaceri).

12. PHP și baze de date: MySQL

- MySQLi este o bibliotecă MySQL avansată pentru gestionarea bazelor de date MySQL.
- MySQLi acceptă concepte obiect și procedurale.
- Creăm un obiect MySQLi nou cu constructorul MySQLi().
- Efectuăm interogări MySQLi folosind metoda **real_query** sau **query** MySQLi object.
- Preluăm rezultatele interogării cu metodele **store_result** sau **use_result** pentru citirea buffered și fără buffer.
- Citim valorile rândurilor rezultatelor interogării MySQLi folosind metode **fetch**.
- Pentru a crea interogări „pregătite”, folosim metoda **prepare** a obiectului MySQLi.
- În interogarea pregătită, parametrii sunt reprezentați printr-un semn de întrebare **?**.

- Fiecare interogare pregătită (obiect Statement) trebuie să aibă parametri alocați. Acești parametri sunt alocați folosind metoda **bind_param**.
- Pentru a activa interogarea tranzacțional, trebuie dezactivată opțiunea autocommit, folosind metoda **autocommit** a obiectului MySQLi.
- O tranzacție activă este acceptată prin metoda **commit** și revocată prin metoda **rollback**.

13. PHP și baze de date: PDO

- Pentru a crea un obiect PDO, este necesar să-i transmitem constructorului un DSN corect.
- O interogare obișnuită, de la care nu avem nevoie de ResultSet, o începem cu metoda PDO **exec**.
- Dacă dorim un ResultSet ca rezultat al interogării, folosim metoda **query** PDO.
- Pentru a formata rezultatele într-un anumit mod, folosim metode fetch în combinație cu constante adecvate.
- Creăm interogări parametrizate, pregătite folosind metoda **prepare** a obiectului PDO.
- Pentru a atribui parametrii interogării pregătite, folosim metoda **bindParam**.
- O tranzacție într-un obiect PDO începe cu metoda **beginTransaction**.
- O tranzacție într-un obiect PDO se termină cu metoda **rollBack** sau **commit**, în funcție de succesul tranzacției.

3. Programare web

14. PHP și controale HTML

- PHP poate fi imbricat în cod HTML.
- PHP poate emite cod HTML.
- Controalele HTML pot fi procesate și emise prin intermediul funcțiilor PHP.
- Plasarea codului HTML în cadrul funcțiilor PHP contribuie la funcționalitate, încapsulare și transparență.
- De asemenea, contribuie la utilizarea claselor și metodelor (încapsulare, transparență, funcționalitate).
- Codarea inline sau codul spaghetti reprezintă procesarea HTML și a codului serverului (PHP) în același document.
- Procesarea codului serverului și HTML în documente separate se numește code behind.

15. Mișcarea datelor prin procesul Request/Response

- Protocolul HTTP este stateless (nu poate stoca starea utilizatorului).
- Este posibilă salvarea stării în procesul Request/Response cu ajutorul unor metode alternative.
- Pentru a salva starea în procesul Request/Response, putem folosi una din mai multe metode (GET/POST, cookie, session).
- GET/POST – nesigur, cookie - mai sigur, session - cel mai sigur.
- GET/POST – datele sunt transferate de la client la server de fiecare dată.
- Cookie – datele sunt la client.
- Session – datele sunt pe server.
- Implicit, sesiunea funcționează cu ajutorul cookie-urilor, dar este posibil să fie gestionată și manual, cu ajutorul codului PHPSISSID.
- Există mai multe metode care conțin o cerere (POST, GET, HEAD, DELETE etc.) în protocolul HTTP.

16. Metode HTTP: GET, POST și procesarea formularului HTML

- Pentru ca controalele HTML să fie afișate pe o pagină web, este necesar să se afle în tagurile <form>. și </form>. Aceste taguri trebuie să înconjoare toate controalele dintr-un formular.
- Tagul <form> conține trei atribute importante pentru procesarea formularelor, și anume – name, action, method.
- atributul Name – se referă la numele întregului formular. Nu este întotdeauna necesar să denumim formularul, dar este considerată o bună practică de programare.
- atributul Action – valoarea acestui atribut reprezintă adresa URL a programului sau a paginii care procesează datele introduse în formular.
- atributul Method – determină modul în care sunt trimise datele. Poate avea valori GET sau POST.
- Metoda GET de trimitere a datelor implică faptul că datele din formular sunt adăugate la pagina URL la care sunt trimise. Aceste date sunt separate de adresa URL printr-un semn de întrebare (?).
- Metoda GET este tot mai puțin folosită ca metodă de transmitere a datelor, având în vedere că există o limită a numărului de caractere care pot fi trimise așa. De asemenea, datele trimise prin metoda GET sunt vizibile în browser, astfel că este amenințată confidențialitatea datelor.
- Metoda POST trimite date către o anumită pagină ca flux de date separat. Acestea nu sunt vizibile pentru browser și nu există o limită pentru numărul de caractere.
- Ori de câte ori este posibil, trebuie utilizată metoda POST pentru a trimite date.
- Pentru a prelua valori din câmpurile de formular, PHP folosește trei tipuri de variabile: $_GET[],$_POST[] și $_REQUEST[].

17. Administrarea stărilor aplicației: Cookies și Session

- Un cookie este o cantitate mică de informații sub forma unei variabile=valoare pe care browserele web o înregistrează local pe calculatorul utilizatorului. După înregistrare, un script PHP poate citi și utiliza aceste informații.
- Cel mai mare dezavantaj în utilizarea cookie-urilor este faptul că utilizatorul le controlează. Deci, este posibil ca utilizatorul să șteargă cookie-urilor de pe calculator. Atunci, un program PHP care se bazează pe date din cookie nu va funcționa.
- Cookie-ul se înregistrează folosind funcția setcookie(). De exemplu: setcookie(!n!city!n!, !n!Paris!n!).
- Toate informațiile stocate de cookie-uri sunt disponibile în programul PHP prin șirul $_COOKIE. De exemplu, echo !n!Your town is !n! .$_COOKIE[!n!city!n!].
- Putem șterge un cookie ștergându-i valoarea. De exemplu: setcookie(!n!city!n!).
- O sesiune este timpul petrecut de un utilizator pe un site, adică timpul de la intrarea utilizatorului pe un site și până când îl părăsește.
- Toate valorile salvate în timpul unei sesiuni se află în șirul $_SESSION.
- O sesiune ar trebui să fie deschisă la începutul fiecărei pagini folosind funcția session_start().
- Sesiunea se închide prin apelarea funcției session_destroy().

18. Securitate în PHP: o scurtă introducere

- Validarea datelor ar trebui să se facă întotdeauna la intrare și la ieșire.
- Este bine să utilizați un set de date predefinit pentru a valida datele din formularele primite.
- Pentru filtrarea și remedierea datelor de intrare și de ieșire, ar trebui utilizate funcțiile proprii sau încorporate (strip_tags, htmlentities, htmlspecialchars).
- Nu trebuie să se lucreze niciodată direct cu variabile superglobale, ci valorile lor sortate ar trebui să fie transmise variabilelor locale.
- Nu trebuie utilizate variabilele nedefinite, neinițializate (de exemplu, if($administrator)... fără a defini mai întâi $administrator=false; sau $administrator=true;).
- Cross-site scripting este execuția de scripturi fără știrea utilizatorului.
- Un script de server nedorit implementat cu comanda include (include_once, require, require_once) funcționează la nivel de server în contextul aplicației și, de aceea, poate fi foarte periculos.
- Nu ar trebui să încărcați niciodată un document cu o instrucțiune include cu parametrii preluați direct, nefiltrați (**include $_GET['page'])**;.
- Cu excepția cazului în care utilizăm în mod explicit încărcarea scriptului de server prin URL, cel mai bine este să lăsați opțiunea **allow_url_fopen** dezactivată.
- CSRF (Cross-site request forgery) este un concept foarte periculos de utilizare greșită a autentificării utilizatorilor de către atacatori.
- Utilizatorii ar trebui să fie întotdeauna verificați cu ajutorul tokenului pe care îl comparăm din parametri și sesiune.
- Atacurile în care atacatorul provoacă un comportament nedorit al bazei de date prin modificarea contextului de interogare prin controalele utilizatorului se numesc atacuri SQL injection.
- Unul dintre cele mai periculoase caractere pentru SQL injection este apostroful '.
- Se recomandă a se utiliza funcția **mysqli_real_escape_string** pentru a gestiona apostroful.
- Când utilizați funcția **mysqli_real_escape_string**, este necesar să aveți o conexiune deschisă la baza de date.
- Este foarte important să nu se permită afișarea erorilor legate de bazele de date, deoarece acestea dezvăluie unui potențial atacator componente de securitate care îi pot fi de folos pentru a crea un atac injection. De asemenea, afișarea erorilor dă impresia de securitate slabă a aplicației.
- Cele mai cunoscute tehnologii pentru atacurile legate de sesiune sunt session hijacking și session fixation.
- Dacă un atacator folosește sesiunea altcuiva pentru a se conecta la un sistem, este vorba de session fixation.
- Preluarea sesiunii altcuiva se numește session hijacking.
- O măsură bună de securitate a sesiunii este regenerarea ID-ului după fiecare acces reușit (funcția **session_regenerate_id**).
- Cookie-urile sunt stocate pe calculatorul clientului.

- Cookie-urile de domeniu se află în variabila JavaScript document.cookie.
- Informațiile confidențiale din cookie-uri trebuie păstrate doar în formă criptată.