



# PHP OOP

Faisal Alsubaie

# WHAT WE DID ?

I developed a PHP-based login system using Object-Oriented Programming (OOP).  
The system supports different user roles (Manager, Developer, Intern), each with a specific view and access level.



# IT WAS DESIGNED TO SIMULATE A REAL-WORLD EMPLOYEE PORTAL WHERE:

01

- Managers can view all employees

02

- Developers can view interns without salaries

03

- Interns can only view their own data



# HOW WE DID IT ?

- I used core OOP principles like inheritance, encapsulation, and polymorphism.
- Each user role is represented as a class that extends the base User class.
- A UserFactory dynamically creates the appropriate object based on the user's role

# THE LOGIN FLOW:

- User logs in from the login page
- The session stores their user object
- Based on their role, the dashboard displays different data
- Manager sees full employee list, Developer sees intern list (without salary), Intern sees only their info



# CLASSES AND THE PURPOSE OF IT

- user:  
Parent class for all users (Manager, Developer, Intern)
- Manager (Inherits from User):  
Represents a manager role with access to all employee data

- Autoloader:  
Uses `spl_autoload_register()` to automatically include class files from the `classes/` directory.
- Data Source:  
`users_data.php` → associative array that holds all mock user records used during login.

- Developer (Inherits from User):  
Represents a developer role with access to intern data only.
- UserFactory:  
Factory class to create the appropriate user object based on their role.

## UI Files:

- `index.php` → login form
- `dashboard.php` → shows user-specific dashboard based on role
- `logout.php` → session destroy
- `includes/employee_table.php` → shows full employees table
- `includes/intern_table.php` → shows interns (without salary)

# CONCLUSION

This project allowed me to deeply understand how to apply Object-Oriented Programming (OOP) in a practical, real-world scenario using PHP.

I learned how to:

- 1- Design reusable and scalable class structures
  - 2- Use inheritance to simplify role-specific logic
  - 3- Implement role-based access control in a secure way
  - 4- Apply the Factory Pattern to dynamically generate objects
- It also helped me understand the importance of:
- Code organization and separation of concerns
  - Encapsulation and hiding sensitive data (like salaries)
  - Building user-specific experiences based on access level



**THANK  
YOU**

