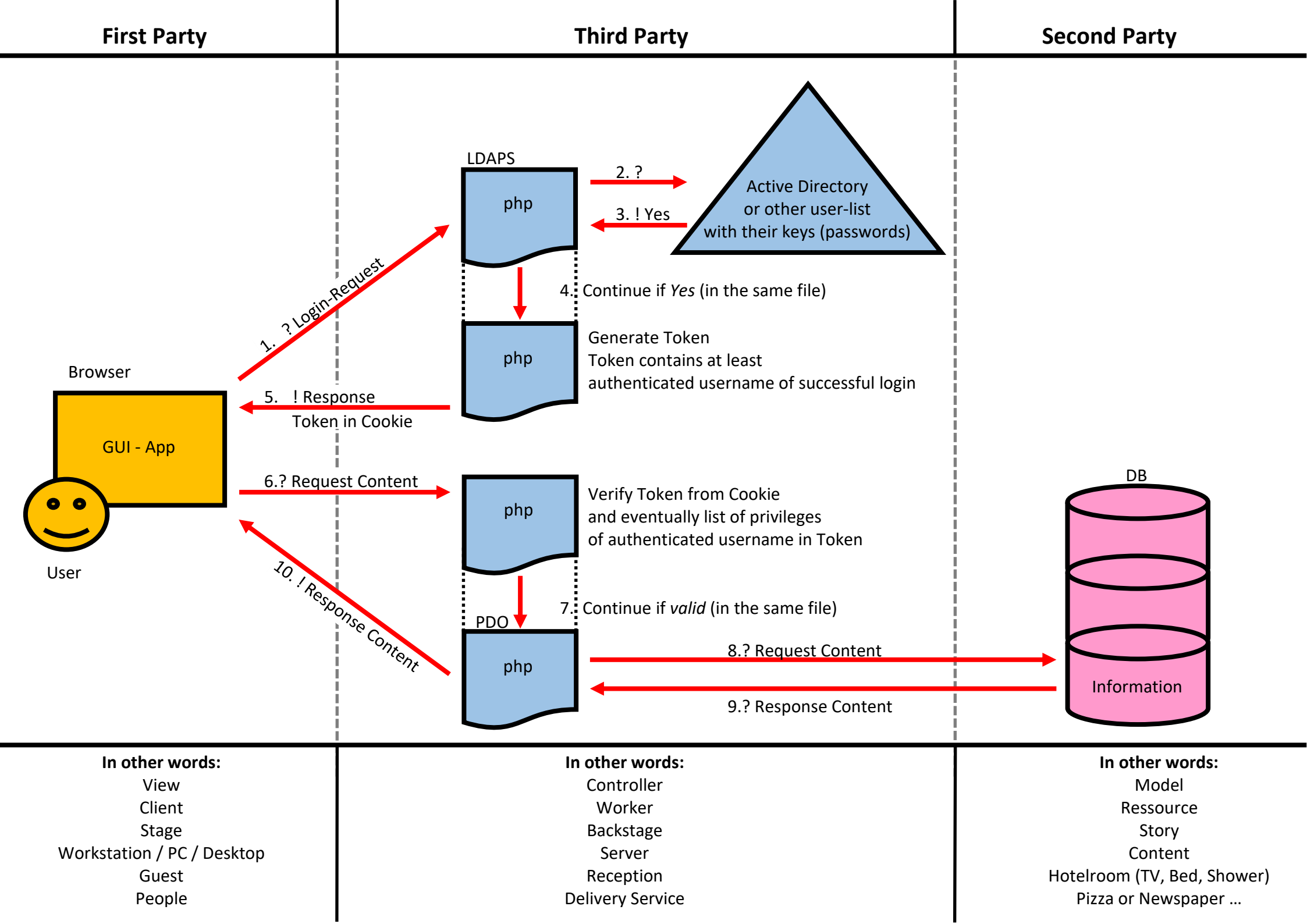
First Party
Third Party
Second Party
LDAPS
php
2. ?
3. ! Yes
Active Directory
or other user-list
with their keys (passwords)
4. Continue if *Yes* (in the same file)
php
Generate Token
Token contains at least
authenticated username of successful login
1. ? Login-Request
Browser
GUI - App
User
5. ! Response
Token in Cookie
6.? Request Content
php
Verify Token from Cookie
and eventually list of privileges
of authenticated username in Token
DB
7. Continue if *valid* (in the same file)
PDO
php
8.? Request Content
9.? Response Content
Information
10. ! Response Content
In other words:
View
Client
Stage
Workstation / PC / Desktop
Guest
People
In other words:
Controller
Worker
Backstage
Server
Reception
Delivery Service
In other words:
Model
Ressource
Story
Content
Hotelroom (TV, Bed, Shower)
Pizza or Newspaper …

Advice for deving authentication in a browser-app

On finding the right decision for the way of authentication in the mass of confusing offers on the net,
this elementary structure and authentication-flow (see above) should be understood.

It doesn't matter if it's

- *third-party trusted authentication*
- *standard network authentication*
- *secure ticketing protocol*
- *SAML*
- *OAuth*
- *KeyCloak*
- *Kerberos*
- *or others,*

or how many other names, signatures, and certificates they invent to be used
and how confusingly complex they are designed and described;
ALL different kinds of authentications (protocols, extensions, frameworks, or apps)
have ONE thing in common. They are grounded on the MVC-pattern.

The Model-View-Controller-Pattern is like a universal STANDARD for all programs or applications.
Even if we don't structure our code accordingly to follow this pattern.
In theory all apps follow this unavoidable pattern-rule.
This is why some say MVC is not a pattern; it is always a fact.

If this simple fact and the default-flow above are understood,
you will find it makes more sense to write your own code for your authentication,
instead of integrating another external dependency that regularly needs updates
and that could lead to hard-to-maintain and inflexible code in your apps.

Because all frameworks have their own strict rules to be followed,
that forces you to restructure your code.
Or it even might be that your new app does not has the right interface/API to be connected.
Then you would need to rewrite your app or still write your own authentication -flow.