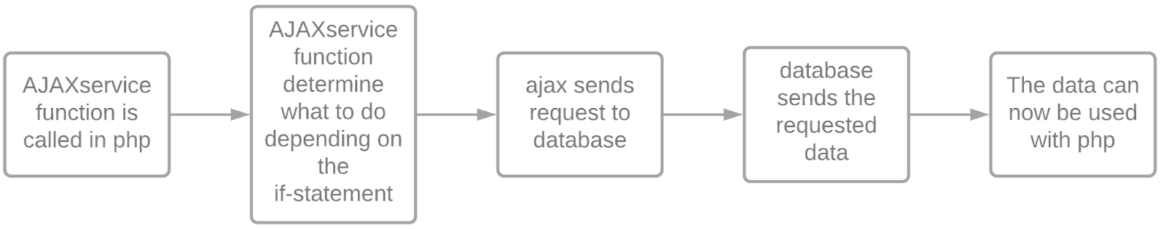

AJAXservice function is called in php
AJAXservice function determine what to do depending on the if-statement
ajax sends request to database
database sends the requested data
The data can now be used with php

Figure 1: Overview of the data flow when using AJAXservice.

Example of the data flow for addSingleUser function in accessed.php:

As shown in *Figure 2,* only the function call should be present on the php page.

```
<input class='submit-button' type='button' value='Add'
onclick='addSingleUser();' />
```

Figure 2: addSingleUser function called by accessed.php when pressing the submit-button.

An example of code is shown in *Figure 3 and Figure 4* how the function works. In *Figure 3* the function prepares the data to be sent through the AJAXservice and this is done by creating the array to store said data. When using or creating a function it is important to write all necessary elements correctly so that the data can be stored properly (these are defined in *Figure 5*). In *Figure 4* we instead use the function to retrieve data from the database.

If you have a form with student or teacher input that needs to be verified, `if (!verifyUserInputForm(newUser)) return;` should be used.

The previous array newUser is then stored in an outer array which is then used later when sending information to the AJAX function.

For the outer array to be able to be sent to the server. It needs to be converted to a string which is done with“`var newUserJSON = JSON.stringify(outerArr);`”.

At this point the data is sufficiently stored and converted correctly to be used when calling the AJAX function.

When calling the AJAX function the first part of the parameter “ADDUSR” is an $opt that is defined in accessservice.php. Here we want to add a user to the database. If left undefined or it is empty, the default is used and is defined under “// Retrieve Information”.

Inside the brackets are the names of the tables that should receive the data we want to send.

The last part “ACCESS” is used to define which part of the AJAXservice we want to use as -shown in *Figure 6*.

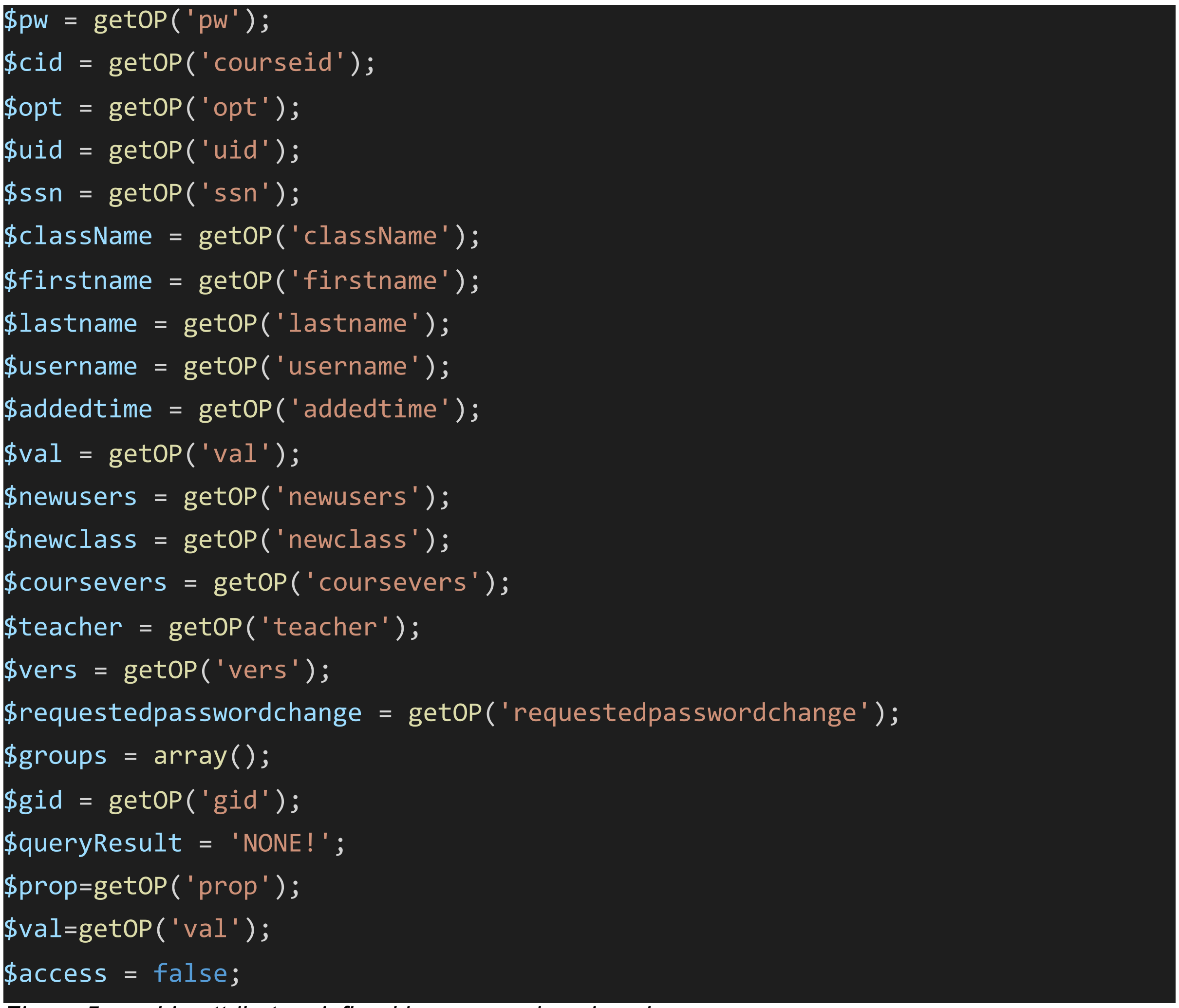
`hideCreateUserPopup();` is used to close the form used to retrieve user input.

```
function addSingleUser() {

    var newUser = new Array();
    newUser.push($("#addSsn").val());
    newUser.push($("#addFirstname").val());
    newUser.push($("#addLastname").val());
    newUser.push($("#addEmail").val());
    newUser.push($("#addCid").val());
    newUser.push($("#addTerm").val());
    newUser.push($("#addPid").val());

    if (!verifyUserInputForm(newUser)) return;
    var outerArr = new Array();
    outerArr.push(newUser);

    var newUserJSON = JSON.stringify(outerArr);
    AJAXService("ADDUSR", {
        courseid: querystring['courseid'],
        newusers: newUserJSON,
        coursevers: querystring['coursevers']
    }, "ACCESS");
    hideCreateUserPopup();
}
```

Figure 3: addSingleUser function in accessed.js calls AJAXService with data.

```
AJAXService("GET", {
        courseid: querystring['courseid'],
        coursevers: querystring['coursevers']
    }, "ACCESS");
```

Figure 4: Example of retrieving data with AJAX from function setup() in access.js



```php
$pw = getOP('pw');
$cid = getOP('courseid');
$opt = getOP('opt');
$uid = getOP('uid');
$ssn = getOP('ssn');
$className = getOP('className');
$firstname = getOP('firstname');
$lastname = getOP('lastname');
$username = getOP('username');
$addedtime = getOP('addedtime');
$val = getOP('val');
$newusers = getOP('newusers');
$newclass = getOP('newclass');
$coursevers = getOP('coursevers');
$teacher = getOP('teacher');
$vers = getOP('vers');
$requestedpasswordchange = getOP('requestedpasswordchange');
$groups = array();
$gid = getOP('gid');
$queryResult = 'NONE!';
$prop=getOP('prop');
$val=getOP('val');
$access = false;
```

Figure 5: usable attributes defined in accessedservice.php

In the AJAXService function there are if-statements that determine what kind that should be used, this is specified as shown in *Figure 6* by the ACCESS keyword. We can also see that the request is going to be sent to accessedservice.php. The type is POST which means what type of HTTP request that should be used. What data and data type is being sent/used can also be seen here.

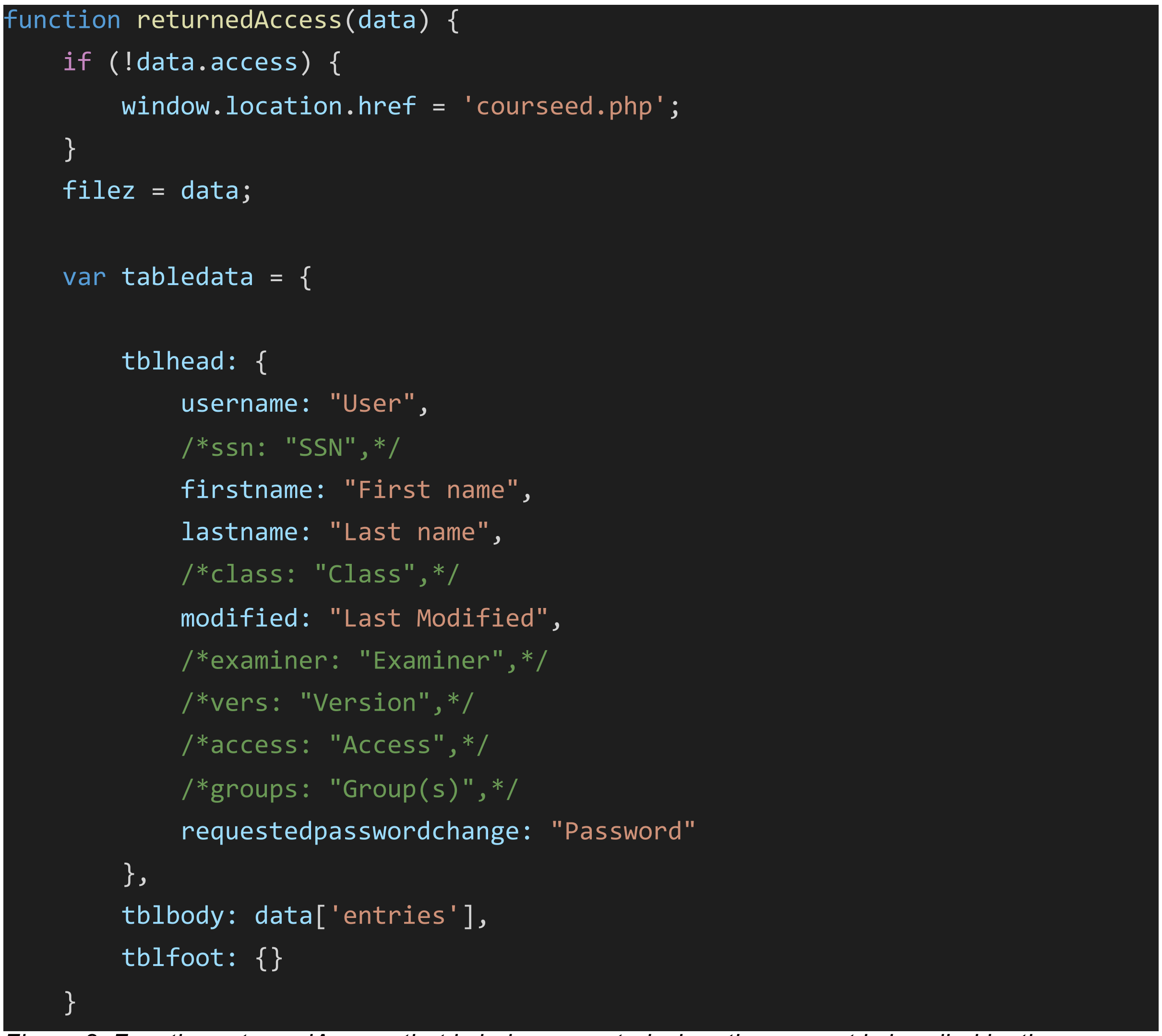
Figure 7 shows a portion of the accessedservice.php, where the MYSQL commands are written for connecting a user to the database.

Success means that when the request to the database succeeds the function returnedAccess (*Figure 8*) is going to be executed.

```
}else if(kind=="ACCESS"){
            $.ajax({
                url: "accessedservice.php",
                type: "POST",
                data: "opt="+opt+para,
                dataType: "json",
                success: returnedAccess
            });
```
Figure 6: Ajax function for kind ACCESS in dugga.js

```
// We have a user, connect to current course
          if($uid!="UNK"){
                        $debug=$regstatus;
                        if($regstatus=="Registrerad"||$regstatus=="UNK"){
                                $stmt = $pdo->prepare("INSERT INTO user_course
(uid, cid, access,vers,vershistory) VALUES(:uid, :cid,'R',:vers,'') ON
DUPLICATE KEY UPDATE vers=:avers, vershistory=CONCAT(vershistory,
CONCAT(:bvers,','))");
                                $stmt->bindParam(':uid', $uid);
                                $stmt->bindParam(':cid', $cid);
                                $stmt->bindParam(':vers', $coursevers);
                                $stmt->bindParam(':avers', $coursevers);
                                $stmt->bindParam(':bvers', $coursevers);

                                // Insert the user into the database.
                                try {
                                        if(!$stmt->execute()) {
                                                $error=$stmt->errorInfo();
                                                $debug.="Error connecting user
to course: ".$error[2];
                                        }
                                }catch(Exception $e) {

                                }
                        }
          }
```
Figure 7: Data being prepared for the request to the database by accessedservice.php.

When the request has been succeeded the function from the “success” row in *Figure 5* is being executed. In this case it is the returnedAccess function in *Figure 8*. In this function we can use the data that was retrieved by the request.



```
function returnedAccess(data) {
    if (!data.access) {
        window.location.href = 'courseed.php';
    }
    filez = data;

    var tabledata = {

        tblhead: {
            username: "User",
            /*ssn: "SSN",*/
            firstname: "First name",
            lastname: "Last name",
            /*class: "Class",*/
            modified: "Last Modified",
            /*examiner: "Examiner",*/
            /*vers: "Version",*/
            /*access: "Access",*/
            /*groups: "Group(s)",*/
            requestedpasswordchange: "Password"
        },
        tblbody: data['entries'],
        tblfoot: {}
    }
```

Figure 8: Function returnedAccess that is being executed when the request is handled by the database.

→

```
    //myTable = undefined;
    var colOrder = ["username",/* "ssn",*/ "firstname", "lastname",
/*"class",*/ "modified", /*"examiner", "vers", "access", "groups",*/
"requestedpasswordchange"]
    if (typeof myTable === "undefined") { // only create a table if none
exists
        myTable = new SortableTable({
            data: tabledata,
            tableElementId: "accessTable",
            filterElementId: "filterOptions",
            renderCellCallback: renderCell,
            renderSortOptionsCallback: renderSortOptions,
            renderColumnFilterCallback: renderColumnFilter,
            rowFilterCallback: rowFilter,
            displayCellEditCallback: displayCellEdit,
            updateCellCallback: updateCellCallback,
            columnOrder: colOrder,
            freezePaneIndex: 4,
            hasRowHighlight: true,
            hasMagicHeadings: false,
            hasCounterColumn: true
        });
        shouldReRender = true;
    }

    if (shouldReRender) {
        shouldReRender = false;
        myTable.renderTable();
    }
}
```
Figure 8: continued.