

# The role of the middle layer in the web environment

---

**Halupecki, Elvis**

**Undergraduate thesis / Završni rad**

**2023**

*Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj:* **Bjelovar University of Applied Sciences / Veleučilište u Bjelovaru**

*Permanent link / Trajna poveznica:* <https://um.nsk.hr/um:nbn:hr:144:551961>

*Rights / Prava:* [In copyright](#) / [Zaštićeno autorskim pravom.](#)

*Download date / Datum preuzimanja:* **2024-05-16**



*Repository / Repozitorij:*

[Repository of Bjelovar University of Applied Sciences - Institutional Repository](#)



BJELOVAR UNIVERSITY OF APPLIED SCIENCES  
UNDERGRADUATE PROFESSIONAL PROGRAMME OF STUDY IN  
COMPUTER SCIENCE

## **The role of the middle layer in the web environment**

Final thesis no. 17/RAČ/2022

Elvis Halupecki

Bjelovar, April 2023



Veleučilište u Bjelovaru  
Trg E. Kvaternika 4, Bjelovar

## 1. DEFINIRANJE TEME ZAVRŠNOG RADA I POVJERENSTVA

Student: **Elvis Halupecki**

JMBAG: 0314019093

Naslov rada (tema): **The role of the middle layer in the web environment**

Područje: **Tehničke znanosti**

Polje: **Računarstvo**

Grana: **Informacijski sustavi**

Mentor: **Tomislav Adamović, mag. ing. el.**

zvanje: **viši predavač**

**Članovi Povjerenstva za ocjenjivanje i obranu završnog rada:**

1. **Ivan Sekovanić, mag. ing. inf. et comm. techn., predsjednik**
2. **Tomislav Adamović, mag. ing. el., mentor**
3. **Krunoslav Husak, dipl. ing. rač., član**

## 2. ZADATAK ZAVRŠNOG RADA BROJ: 17/RAČ/2022

As part of the final thesis, it is necessary to:

1. Implement session and header handling on the middle layer in PHP
2. Create .docx and .xlsx templates and populate data for generating reports through the middle layer in PHP
3. Develop a function for sending emails in PHP
4. Process a request on the middle layer in PHP using JSON for communication between layers
5. Embed a logging system on the screen, in a file, and in a database using OOP concepts in PHP
6. Implement a call to a web service for retrieving an image in PHP

Datum: 23.09.2022. godine

Mentor: **Tomislav Adamović, mag. ing. el.**



### *Acknowledgements*

I would like to thank all lecturers from the college of applied sciences in Bjelovar who ensured the best conditions for studying and developing my skills. I would also like to thank my family who supported me throughout all these years. I would also like to express my deepest appreciation to my mentor Tomislav Adamović who helped me in the creation of this final thesis.

# Table of contents

<b>1. Introduction</b>	<b>1</b>
<b>2. The role of the middle layer in the web development</b>	<b>2</b>
2.1 <i>Implement session and header handling on the middle layer in PHP</i>	2
2.1.1 Session	2
2.1.2 Header	3
2.2 <i>Create .docx and .xlsx templates and populate data for generating reports through the middle layer in PHP</i>	4
2.2.1 PHPOFFICE	4
2.2.2 PHPSpreadsheet	4
2.2.3 PHPWord	9
2.3 <i>Develop a function for sending emails in PHP</i>	12
2.4 <i>Process a request on the middle layer in PHP using JSON for communication between layers</i>	13
2.4.1 Sending data using AJAX on the middle layer	13
2.4.2 Receiving and processing data in router.php file	14
2.5 <i>Embed a logging system on the screen, in a file, and in a database using OOP concepts in PHP</i>	15
2.5.1 Comparison of PDO and MySQLi extensions	17
2.5.2 Purpose of PHP ORM libraries	19
2.5.3 Login function	21
2.5.4 Logging into a file	24
2.5.5 Logging in using JWT	25
2.6 <i>Implement a call to a web service for retrieving an image in PHP</i>	29
2.6.1 Call to a web service using the Curl library	29
<b>3. CONCLUSION</b>	<b>32</b>
<b>4. LITERATURE</b>	<b>33</b>
<b>5. ABBREVIATIONS AND ACRONYMS</b>	<b>34</b>
<b>6. ABSTRACT</b>	<b>35</b>
<b>7. SAŽETAK</b>	<b>36</b>

## **1. Introduction**

Every web application consists of 3 layers. Frontend, middle layer (also called middle layer) and backend. The role of frontend is to show information in an organised and orderly manner. Backend is used for saving data received from a user, and returning data requested by a user. Middleware can have multiple usages. The most common ones are converting data received from frontend (eg. Converting a JSON array into an ordinary PHP array) and for authorization and authentication, which checks if a user is allowed to see or edit certain information on the website. Middleware is also used for saving pictures and videos on the server, and for data validation so that the data received from frontend is correct and free of any errors that might negatively affect the database.

## 2. The role of the middle layer in the web development

The paper is based on a project in which users log in to a website where they sign up for a programming course. For ordinary users, the website consists of a registration form, login form and a page where they request a barcode for a payment slip which is used to pay for the course. Users with administrator rights have a page on which they can request a statistics report for all users in the database which will be generated in excel. They also have access to a page where they can generate consent papers for each user.

### 2.1 Implement session and header handling on the middle layer in PHP

The usage of sessions and headers is a standard practice when developing a webpage because of their advantages. Sessions and headers will be used in PHP which is the main programming language in this project.

#### 2.1.1 Session

Sessions are used to store data about the user on the server which is the opposite of cookies, which are stored locally. When a user logs into his account on the website using his username and password, a session is created with a name which consists of a random combination of numbers and letters. That session is then used across all web pages in the same domain in order not to ask for credentials each time a user changes a webpage.

Session can also save some data about the user (eg. his username) which can be used to personalize web pages for each individual user. In the program code 2.1 the function `session_start()` creates a new session. In this example the function is called if the username and password entered by the user pass the verification process, which then fills the `$_SESSION['login']` variable with the username. The advantage of having user-related data in a session is that this data can be used to personalize the webpage without the need to call the database every time the user changes a webpage.

---

```
if(isset($_POST['submit'])) {  
    include('DbFunction.php');  
    $obj=new DbFunction();  
    $success= $obj>login($_POST['username'],$_POST['password']);  
  
    if($success) {  
        session_start();  
        $_SESSION['login'] = $_POST['username'];  
  
        header ( 'location:../generatePayslip.php' );  
    }  
}
```

---

*Program code 2.1 Session and header*

### 2.1.2 Header

Headers can be used to determine to which web page the user will be redirected to after some interaction on it. In program code 2.1, the `header()` function is used to redirect the user to `generatePayslip.php`. Also, headers can be used to determine which type of data will be returned to the user. In program code 2.2, while generating an Excel table, 2 parameters are added to the header function, „*Content-Disposition: attachment*“ which informs the browser that it should expect a file that will be downloaded to the computer, and „filename“ which determines the name of the file that will be downloaded.

---

```
// Gives information that the file has to be  
downloaded, and the filename  
  
header('Content-Disposition:  
attachment;filename="statistics.xlsx"');
```

---

*Program code 2.2 Using a header to download a file*



## 2.2 Create .docx and .xlsx templates and populate data for generating reports through the middle layer in PHP

Templates are generated with the PHPOFFICE library. A template in excel is used to display user statistics from the database. It will show a table of users, selected groups and terms, and whether they paid for the programming course or not. Next to the table, there will be a graph that displays the number of users in each group and term for easier data analysis. The template generated in Word consists of a consent form for processing personal information, where the user's name, surname and current date will be automatically filled in.

### 2.2.1 PHPOFFICE

PhpOffice is a group of libraries used to generate files in the PHP programming language. PhpOffice has the capabilities to create and edit files from the popular Microsoft applications, some of which are Word, Visio, Excel and PowerPoint. In this paper, the PHPWord library is used for editing .docx templates and the PHPSpreadsheet, which is used for creating and populating .xlsx templates. Before using the aforementioned libraries, they need to be installed with PHP Composer.

### 2.2.2 PHPSpreadsheet

With PHPSpreadsheet it is possible to create spreadsheets including most of the additional capabilities which can be created in Excel, like creating graphs and diagrams, selecting different fonts, calculating the average and others. Creating .xlsx files starts with creating a spreadsheet and choosing the worksheet. In this case, a new spreadsheet will be created which will have statistics of users in the database. In program code 2.3, firstly a new spreadsheet is created and the current worksheet is fetched. Then, on program code 2.4, the header of the table is created which will contain user data collected from the database. method *setCellValue()* takes the cell label and the text that should be in it as the input.

---

```
$spreadsheet = new Spreadsheet();  
$worksheet = $spreadsheet->  
getActiveSheet();
```

---

*Program code 2.3 Creation of a new spreadsheet*

---

```

$spreadsheet->getActiveSheet()
    ->setCellValue('A2','ID')
    ->setCellValue('B2','Email')
    ->setCellValue('C2','Name')
    ->setCellValue('D2','Surname')
    ->setCellValue('E2','Sex')
    ->setCellValue('F2','Group')
    ->setCellValue('G2','Term')
    ->setCellValue('H2','Payment status');

```

---

*Program code 2.4 Changing the values of the cells*

To change the text style in the whole document, or the font, color, size and so on, the *getDefaultStyle()* method is used, which is called from the object that contains the spreadsheet. In the program code 2.5, font „Arial“ and font size are set, while the result can be seen on picture 1. The result is used with the method *setCellValue()*.

---

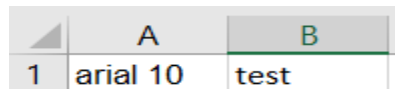
```

$spreadsheet->getDefaultStyle()
->getFont()
->setName('Arial')
->setSize(10);

```

---

*Program code 2.5 Changing of the font in the whole xlsx document*



	A	B
1	arial 10	test

*Picture 1 Text layout after changing the text style*

If the change of font is needed only in a specific cell, method *getStyle()* is used with the label of the cell as the input parameter. From there, methods *getFont()* and *setSize()* are called, *getFont()* fetches the text, while the *setSize()* sets the fetched text to the desired size. Program code 2.6 shows the code for changing the font size of one cell, while picture 2 shows the result.

---

```

$spreadsheet->getActiveSheet() -
>getStyle('A2')->getFont()->setSize(20);

```

---

*Program code 2.6 Methods used for changing the size of the font in a cell*

	A	B
1	arial 10	test
2	arial 20	

Picture 2 Result after changing the font size in a cell.

In order to have a long text and be able to read it, cells have to be merged. The `mergeCells()` method with the input parameter of the starting cell and the ending cell creates the desired result. Program code 2.7 shows the method for merging cells from cell A1 to cell F1, and the picture 3 shows the result.

---

```
$spreadsheet->getActiveSheet() -
>mergeCells("A1:F1");
```

---

Program code 2.7 Method used to merge cells

	A	B	C	D	E	F
1	Users					

Picture 3 Result of using the `mergeCells()` method

To use the method for the alignment of text inside of a cell, it's needed to select a cell and it's characteristics with the method `getStyle()` with the cell label as the input parameter. After fetching the cell, it is required to call the `getAlignment()` method which will get the current alignment of the cell, and then call either the `setHorizontal()` or `setVertical()` method, depending on the direction in which the alignment is needed.

Each of the mentioned methods can receive several predefined parameters that will align the text to the desired side. In this case, the parameter `HORIZONTAL_RIGHT` will align the text horizontally to the right side. Program code 2.8 shows the use of the text alignment method, while picture 4 shows the result.

---

```
$spreadsheet->getActiveSheet() -
>getStyle('A1')->getAlignment() -
>setHorizontal(Alignment::HORIZONTAL_RIGHT);
```

---

Program code 2.8 Method used for aligning the text inside of a cell

	A	B	C	D	E	F
1						Users

Picture 4 Result of the `setHorizontal()` method

For the creation of a diagram, the following data is needed: values that will be shown, labels, legend, and the height and width of the diagram. Firstly, an array with 4 arrays inside of it will be created, which will represent groups and the values shown on the diagram. In addition to the names of the groups, the array will also contain the identification number and the number of users registered for each term, as shown in program code 2.9.

---

```
//Id, Morning, Afternoon, Group
$groups = array (
    array(1,0,0,"Youngster"),
    array(2,0,0,"Junior"),
    array(3,0,0,"Pre-Teen"),
    array(4,0,0,"Teen")
);
```

---

*Program code 2.9 Array with groups*

Iterating through all users will fill the specified array, considering which group and term the user belongs to. The data will be printed into cells on the spreadsheet. Values from the cells will then be collected and saved into the *chartData* variable, as shown in program code 2.10. *DataSetValues* is a data type which is exclusively used in the creation of a diagram and that contains information about the range up to which the values on the spreadsheet should be read. In this case, there are 2 *DataSetValues* data types because there are 2 groups that the users can choose from, in the morning and in the afternoon.

---

```
$chartData = [
    new
    DataSetValues(DataSeriesValues::DATASERIES_TYPE_NUMBER
        , 'Worksheet!$P$19:$P$22', null, 2),
    new
    DataSetValues(DataSeriesValues::DATASERIES_TYPE_NUMBER
        , 'Worksheet!$Q$19:$Q$22', null, 2),
];
```

---

*Program code 2.10 Data type that is used to show values on the diagram*

Creation of the legend, naming of the diagram and adding text description to the x-axis can be seen in program code 2.11.

---

```

//legend
$legend = new
ChartLegend(ChartLegend::POSITION_RIGHT
, null, false);

//title
$title = new Title('Users');

// x-axis name
$xAxisLabel = new Title('Grupe');

```

---

Program code 2.11 Function for creating the legend, title and the values on the x-axis

By combining all of the data, a diagram is created, and the entire file is created using the *IOFactory::createWriter()* function, as shown in program code 2.12. Before actually saving and downloading the file, it is necessary to call the *setIncludeCharts()* method with the parameter „true“ in order to display the created diagrams. Saving and downloading is initiated using the *save()* method with the parameter „php://output“ which specifies the location where the file will be saved. In this case, it will be the user's device.

---

```

$writer =
IOFactory::createWriter($spreadsheet, 'Xlsx');

//Enable diagrams
$writer->setIncludeCharts(true);

//Saving the excel file
$writer->save('php://output');

```

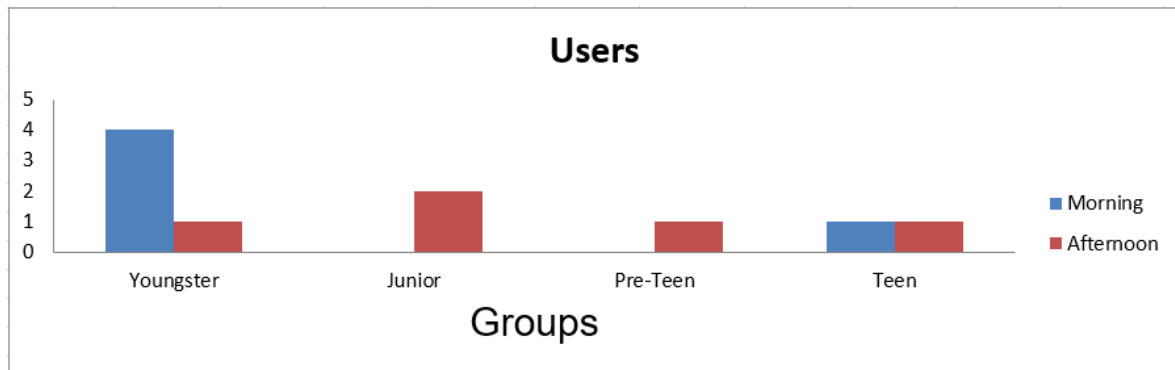
---

Program code 2.12 Saving the excel file

The final result can be seen on picture 5, which shows a table with users and their data, while picture 6 shows the generated diagram.

ID	Email	Name	Surname	Sex	Group	Term	Paymei
1	<a href="mailto:test1@gmail.com">test1@gmail.com</a>	Ivan	mek	Male	Youngster	Morning	Not paid
2	<a href="mailto:test2@gmail.com">test2@gmail.com</a>	Marija	dasda	Female	Youngster	Morning	Paid
3	<a href="mailto:test3@gmail.com">test3@gmail.com</a>	Sara	Maša	Male	Youngster	Morning	Paid
6	<a href="mailto:ttest4@gmail.com">ttest4@gmail.com</a>	Slavka	sad	Female	Youngster	Morning	Not paid
7	<a href="mailto:test5@gmail.com">test5@gmail.com</a>	Goran	test	Male	Junior	Afternoon	Not paid
8	<a href="mailto:test6@gmail.com">test6@gmail.com</a>	Ivan	test	Male	Pre-Teen	Afternoon	Not paid
9	<a href="mailto:ehalupecki@gmail.com">ehalupecki@gmail.com</a>	Ivona	sarko	Female	Teen	Morning	Not paid
10	<a href="mailto:test7@gmail.com">test7@gmail.com</a>	marko	jarko	Male	Teen	Afternoon	Paid
11	<a href="mailto:test8@gmail.com">test8@gmail.com</a>	Karla	tet	Female	Junior	Afternoon	Not paid
12	<a href="mailto:test9@gmail.com">test9@gmail.com</a>	Ivan	Kralic	Male	Youngster	Afternoon	Paid
							40%

Picture 5 Table with users and their data



Picture 6 Diagram that shows the users in each group and term

### 2.2.3 PHPWord

PHPWord is used to create and format text documents. There are options to add images, titles, subtitles, charts and other objects. In addition to creating documents, it is possible to edit and fill document templates. For the creation of a new document, firstly a PhpWord object has to be put into a variable as shown in program code 2.13. The object can now use other methods. In the next row, the method *addsection()* is called, which creates a new section in the document. The new section has to be saved in a new variable in order to be manipulated with, i.e. to be able to add text or new objects.

---

```
$phpWord = new
\PhpOffice\PhpWord\PhpWord();

$section = $phpWord-
>addSection();
```

---

Program code 2.13 Creation of a new word document and a section

Adding text to a section is achieved by calling the *addText()* method with the new text as the input parameter. In program code 2.14, an example of the method call is shown, while on picture 7 the result can be seen.

---

```
$section->addText(
    "This is a text");
```

---

Program code 2.14 Adding text to a section

## This is a text

*Picture 7 Result of the addText() method*

To add different fonts, sizes and colors of letters, a new variable must be created that will include all the data about how the text should look like. In program code 2.15, a new variable called *\$textLayout* is created which is then used in the *addFontStyle* method.

In the mentioned method, 2 input parameters are sent, the variable name and the array of values that are going to be changed. In this case, the font, size and color of letters are changed, including the bolding of the text. The font will be „Broadway“, the size of the letters will be 10, the color will be #2a54fa, which is a shade of blue.

---

```
$textLayout = 'test';
$pdfWord->addFontStyle( $textLayout
    array('name' => 'Broadway', 'size' => 15,
    'color' => '#2a54fa', 'bold' => true)
);
$section->addText(
    "This is a text",
    $textLayout
);
```

---

*Program code 2.15 Adding parameters to create the desired text layout*

Now, when the method *addText()* is called, the second parameter will be the new variable used to add all the changes to the text style. The result can be seen on picture 8.

## This is a text

*Picture 8 Result of the AddText method with additional parameters*

The most important part of generating word files is the replacement of the placeholders. Each placeholder will be changed to a value unique for each user, such as his name, surname or date of birth.

The format used for placeholders inside of the PhpWord library is **\${variableName}**, which can then be replaced using the *setValue(variableName, newValue)* method. The example of the method can be seen on picture 2.16 where a placeholder called „name“ will be replaced with the name of the current user. User's name is fetched from the global variable called *\$\_SESSION['name']* which saves the user's name when he logs in.

---

```
$phpWord->setValue('name',  
$_SESSION['name']);
```

---

*Program code 2.16 Method used to replace the placeholders*

On picture 9, a document with placeholders is shown, while on picture 10, the same document is shown after the change.

**Consent**  
For processing personal data

As a user of our services, you \${name} \${surname}, on \${date}, agree to the processing of personal data collected during the registration on the website.

-----

Date: \${date}

---

*Signature*

*Picture 9 Document before replacing the placeholders*

**Consent**  
For processing personal data

As a user of our services, you Elvis Halupecki, on 04.04.2023, agree to the processing of personal data collected during the registration on the website.

-----

Date: 04.04.2023

---

*Signature*

*Picture 10 Document after replacing the placeholders*



## 2.3 Develop a function for sending emails in PHP

The process of sending emails is done with the PHPMailer library. In order to be able to send an email, it is required to have an email server which will do the sending. Before sending the email, it is necessary to enter the hostname, username, password and the TCP port through which the email will be sent. In this example, a virtual email server will be used which will receive emails sent from the website. In program code 3.1 shows the assignation of all the necessary information for sending emails, including the emails from the sender and receiver. If needed, additional receivers can be added. If the parameter „true“ is added to the *isHTML()* method, instead of a simple text, the email can consist of elements used in a website, like alignments, images, links and backgrounds.

---

```
$mail = new PHPMailer();
$mail->isSMTP();
$mail->Host = 'smtp.mailtrap.io';
$mail->SMTPAuth = true;
$mail->Username = 'f8f5a789f64c63';
$mail->Password = '213453ef7b9cae';
$mail->SMTPSecure = 'tls';
$mail->Port = 2525;
$mail->setFrom('radionica@gmail.com',
'Radionica');
$mail->addAddress('ivica@ivic.net', 'Ivica Ivic');
$mail->isHTML(true);
```

---

*Program code 3.1 Settings needed for sending an email*

In the program code 3.2, a title, text and picture is added to the email. Variable *\$username* is used to fetch the saved barcode with the name of the user that requested it.

---

```
$mail->Subject = "Barcode for course payslip";
$mail->addEmbeddedImage($username . '.png',
'image_cid');
$mail->Body = ' Barcode
of a payslip';
$mail->AltBody = 'Barcode for a payslip';
```

---

*Program code 3.2 Title and the body of the email*

## 2.4 Process a request on the middle layer in PHP using JSON for communication between layers

jQuery AJAX will be used to process requests on the frontend, which is used to receive and send data without needing to reload the page after each request. When sending data, the data will be converted into JSON format from which the necessary data will be processed later.

JSON (JavaScript Object Notation) is a text format used to store and send data. It is very fast and easy to use and read, which is why it is most often used when communicating with a website. An example of a JSON object can be seen in program code 4.1.

---

```
{
    "test": {
        "name": "test"
    }
}
```

---

*Program code 4.1 Example of the JSON object*

JSON consists of a field name and a value inside of that field. Also, it is possible to declare arrays or more fields inside of one JSON field. Instead of JSON, XML(Extensible Markup Language) can also be used, which is older than JSON. XML provides more options when sending the data and it has better security, but it is also slower and harder to read. JSON communication will be used here because of its speed and simplicity.

### 2.4.1 Sending data using AJAX on the middle layer

In program code 4.2, a call towards the middleware is shown in which the user wants to log into the website. After clicking on the log in button, middleware sends the username, the password and the procedure that the user called. The user will then be redirected to the assigned webpage if everything went successfully.

---

```

$('#submit').on('click', function () {

    $.ajax({
        url: 'router.php',
        type: "POST",
        data: {
            function: 'login',
            username: $('#username').val(),
            password: $('#password').val()
        },

        success: function(result) {

            window.location = "generatePayslip.php";

        }
    });
});

```

---

*Program code 4.2 AJAX call towards the middleware*

#### **2.4.2 Receiving and processing data in router.php file**

The data sent by AJAX will be delivered to router.php file as an object and will be sorted and redirected depending on the requested procedure. First, it will check by which method the data was requested and according to it, will be processed with the *json\_encode()* function. The mentioned function converts the received data into JSON for easier processing. After the conversion into JSON, by calling the function *json\_decode()* without any additional parameters, the JSON is converted into an object, while with the „true“ parameter, the JSON is converted into an associative array. The mentioned function is here called with the „true“ parameter in order to read the values from the associative array as it is shown in the program code 4.3.

---

```

switch ($_SERVER['REQUEST_METHOD']) {
    case 'POST':

        $injson = json_encode($_POST);

        break;
    case 'GET':

        $injson = json_encode($_GET);
        break;
    default:
        echo $request_err;
        return;
}
$in_obj = json_decode($injson, true);

```

---

*Program code 4.3 Checking the request method*

## 2.5 Embed a logging system on the screen, in a file, and in a database using OOP concepts in PHP

MySQL database is used for storing data, while the MySQLi PHP extension is used to perform all the necessary operations on the data. It provides an object-oriented interface that makes it much easier to use and manage the data going to and from the database.

The DbFunction file is used to call the functions, which contains the DbFunction class that contains all the functions. The class and some of the function declarations are shown in program code 5.1

---

```
Class DbFunction{

    function registerUser($name,$surname,$usernamePost,
    $passwordPost,$passwordConfirm) {

    }

    function login($usernamePost, $passwordPost){
    }

    function createExcelFile(){
    }

}
```

---

*Program code 5.1 DbFunction class and function declarations*

In order to use the *DbFunction* class, it's required to include the *Database.php* file which contains all the data needed to use the database. Using the *require()* function with the file location as the input parameter, includes the file so that it can be used. Program code 5.2 shows the requiring of the *Database.php* file.

---

```
require('Database.php');
```

---

*Program code 5.2 Requiring of the Database.php file*

Database.php has the data needed to access the database. It consists of the IP address of the database, username and password which are used to access the database and the name of the database. The values needed to access the database are shown in program code 5.3. In this example, the variable *\$\_password* is left empty because the password is not needed since the database is on a local computer.

---

```

private $_connection;
private static $_instance;

private $_host = „localhost:3306“;
private $_username = „root“;
private $_password = „“;
private $_database = „project“;

```

---

*Program code 5.3 Values needed to access the database*

As seen in the program code 5.3, in addition to the database access data, there are two more variables, the variable `$_connection` and the variable `$_instance`. With the variable `$_connection`, a connection will be created inside of the constructor of the Database class. The constructor will be called every time when an object is created from the mentioned class. In program code 5.4, a constructor from the Database class is shown. In PHP, the *construct* variable must start with 2 underscores. A mysqli connection is saved in the `$_connection` variable which has all the data needed to connect to the database. If an error occurs, for example, a password is incorrect or the database is unavailable, the error and error code will be shown.

---

```

public function __construct() {
    $this->_connection = new mysqli($this->_host,
    $this->_username,    $this->_password, $this->_database);

    // Error
    if(mysqli_connect_error()) {
        trigger_error("Not possible to connect to
the database, error code: " . mysqli_connect_error(),
        E_USER_ERROR);
    }
}

```

---

*Program code 5.4 Constructor in the Database class*

Program code 5.5 shows the role of the `$_instance` variable. It is used to check and save an instance, if one currently does not exist. The reason why it is used is to reduce unnecessary connections to the database in order to increase the performance of the website.

---

```

public static function getInstance() {
    if(!self::$_instance) { // Creating instance
        self::$_instance = new self();
    }
    return self::$_instance;
}

```

---

*Program code 5.5 Checking and creating a new instance of the database class*

Inside of the *Database* class, there is also the *getConnection()* function which has to be called before every process which requires the database. The mentioned function will enable the connection to the database. The function is shown in program code 5.6

---

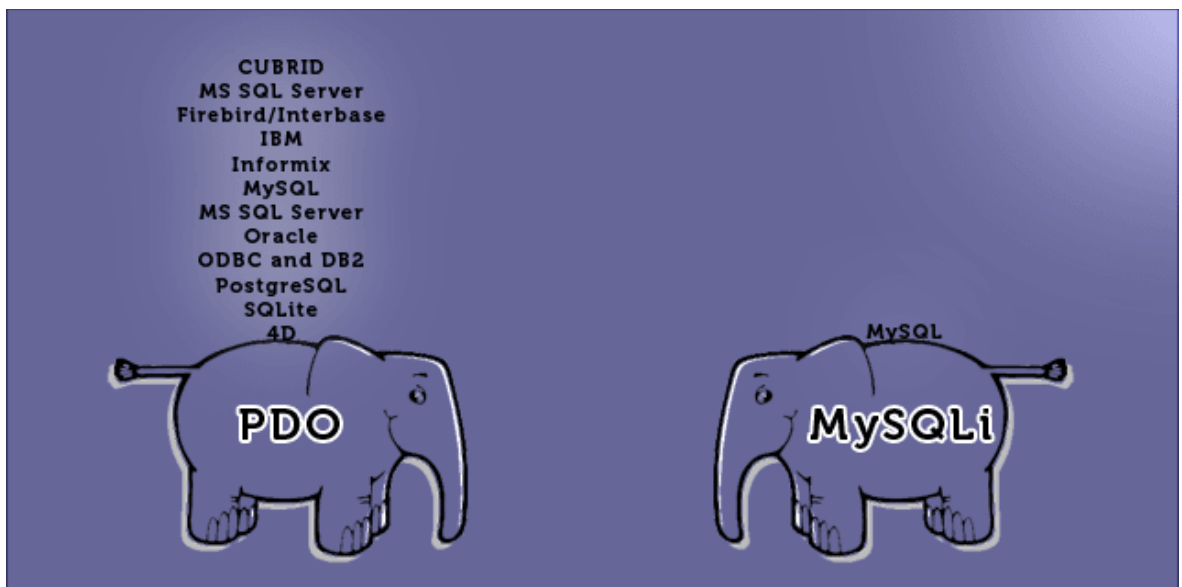
```
public function getConnection() {  
    return $this->_connection;  
}
```

---

*Programski kod 5.6 getConnection function*

### 2.5.1 Comparison of PDO and MySQLi extensions

The alternative to *MySQLi* extension is PDO(PHP Data Objects) which also provides an object-oriented interface to manage the database. The main difference between PDO and *MySQLi* extension is that *MySQLi* can only use the MySQL database, while PDO can be used with 12 different databases, including *MySQL*.



*Picture 11 Database types supported by PDO and MySQLi*

Both extensions offer the option to use prepared queries which have some advantages. The security advantage of prepared queries is the prevention of SQL Injection with prior checking and cleaning of inputted data. The performance advantage of prepared queries is the speed of execution.

A security problem is shown where a malicious user would pass SQL code as an input parameter instead of an email when logging into the website. In program code 5.7 there is a potential query to the database that would cause SQL injection i.e. delete everything from the user table if it reached the database. In program code 5.8, the query in which the preceding text would be inserted is shown.

---

```
$_POST['code'] = "'; DELETE FROM users; /*";
```

---

*Program code 5.7 Query that would cause a SQL injection*

---

```
SELECT email, password FROM users WHERE email =
```

---

*Program code 5.8 Query that expects an email*

The problem arises when the specified text reaches the database. The database misinterprets the entered text, and, instead of searching for the user's email, it executes the delete command. For this reason, prepared queries are used to prevent the mentioned problem.

Prepared queries consist of 2 parts: preparation and execution. In the preparation part, a query is created towards the database that will be used as a template. The query will not have any data in itself, instead it will just be used to prepare the database for what kind of a query it will receive in the near future, which will significantly increase the speed of execution. Program code 5.9 shows an example of a prepared query in *mysqli* which will be sent to the database in the preparation stage. The question mark represents the value that will be filled in during the execution stage.

---

```
$mysqli->prepare("SELECT email, password FROM users WHERE  
email = ? AND password = ?")
```

---

*Program code 5.9 Prepared query in MySQLi*

When it comes to the PDO prepared query, the code syntax is mostly the same except instead of using a question mark, a colon is used with the name of the variable as shown in program code 5.10.

---

```
$pdo->prepare( SELECT email, password FROM users WHERE email =  
:email AND password = :password );
```

---

*Program code 5.10 Prepared query in PDO*

In the execution stage, data is sent into the previously prepared template query. There, with the use of the *mysqli bind\_param* method, the question marks are replaced with values after which the *execute* method executes the query. First value represents the type of the data that is being sent. „i“ represents an integer, „s“ represents a string, „d“ represents a number with a decimal point, also known as float. In this case, string will be used, as the email can consist of letters and numbers. In PDO, the temporary values are immediately passed to the execute method. They are passed in an array of values with the format „:variable => value“. Example of query execution in *mysqli* is shown in program code 5.11, while the example of PDO is shown in program code 5.12.

---

```
$mysqli->bind_param('s', $email);  
$mysqli->execute();
```

---

*Program code 5.11 Execution of the prepared query in MySQLi*

---

```
$pdo->execute(array(':email' => $email));
```

---

*Program code 5.12 Execution of the prepared query in PDO*

Both extensions have a large amount of examples and how to use them online. MySQLi gets more frequent updates due to the support of only one database type, which is why it is recommended to use if MySQL database is going to be used. But the decision on which extension to use is on the developer who will decide which methods are needed and if the project will grow in the future, which would make it necessary to switch to a different database type.

### 2.5.2 Purpose of PHP ORM libraries

For easier management of the database, ORM (Object-Relational Mapping) libraries can be used. They offer object-oriented management of the data in the database, which will greatly reduce the number of manually written SQL commands, resulting in program code that is more readable and less prone to errors.

Each table that exists in the database must have its own class if it is to be used by ORM. The class will have attributes equal to the ones in the table. In program code 5.13, an example of creating a class with Doctrine ORM library is shown. First line indicates the location of the Doctrine library and allows the library to be called. Each line which starts with `@ORM` indicates to the library that it contains data that the library needs to read. `@ORM\Entity` indicates that the library will be an entity, while `@ORM\Table(name = users)` indicates the table by which the entity will be created.

In the class „User“, the specifications of each field are defined. `@ORM\Column(type = „integer“)` will indicate that the field will be an integer while `ORM\Id` indicates that the field will be an identification number and will automatically make the field unique. `ORM\GeneratedValue(strategy = „AUTO“)` sets the increment to 1 which will result in increasing the id field by 1 for each new record. Another attribute that this class has is email, which will be a string of a maximum of 50 characters.

---

```
use Doctrine\ORM\Mapping as ORM;
/**
 * @ORM\Entity
 * @ORM\Table(name = "users")
 */
class User {
    /**
     * @ORM\Column(type = "integer")
     * @ORM\Id
     * @ORM\GeneratedValue(strategy = "AUTO")
     */
    private $id;

    /**
     * @ORM\Column(type = "string", length = 50)
     */
    private $email;
}
```

---

Program code 5.13 Creation of a class with Doctrine library



Functions that define what the class can do, are then added in the same class. In this example, function *getEmail()* will fetch a user's email, while *setEmail()* will set the user's email. An example is shown in program code 5.14.

---

```
public function getEmail(): string
{
    return $this->email;
}

public function setEmail(string $email): void
{
    $this->email = $email;
}
```

---

*Program code 5.14 Example function in a class created with the Doctrine library*

After adding the attributes and functions to the class, the same can now be used. Firstly, an object has to be created. Program code 5.15 shows the creation of an object with the command *new User()*; and setting the email by calling the *setEmail(\$\_POST['email'])*, where *\$\_POST['email']* represents the user's email collected from the form.

---

```
$newUser = new User();
$newUser->setEmail($_POST['email']);
```

---

*Program code 5.15 Creation of a new object and setting an attribute*

In order to save the created object to the database, it is necessary to call the *entityManager*. *EntityManager* is a built-in class that handles all functions that involve objects and the database. In program code 5.16, the *entityManager* is called with the *->persist(\$newUser)* function, which informs the *entityManager* that a new object is to be added to the database. Using the *flush()* function, all the actions that have been previously called with the *persist()* function, will be executed and saved to the database

---

```
$entityManager->persist($newUser);
$entityManager->flush();
```

---

*Program code 5.16 Saving an object to a database*

ORM libraries have not been used in this paper because there have been only a few SQL queries. Because of that, using ORM in this project would make it more complicated and would potentially harm the current performance, as it is another library that needs to be added to the project.

### **2.5.3 Login function**

When logging in, the login function is called which gets the input data from the user as its input parameters, in this case it is the username and the password. They are saved into variables named *usernamePost* and *passwordPost*. In program code 5.17, the first step is to retrieve the instance and establish a connection to the database. After that, the username and password are checked to ensure that they are not empty. If they are not, the

data is passed to the trim() function, which removes spaces and other non-alphanumeric characters. The reason why this function is used is to increase the security of the database.

---

```
function login($usernamePost, $passwordPost){

    $db = Database::getInstance();
    $mysqli = $db->getConnection();

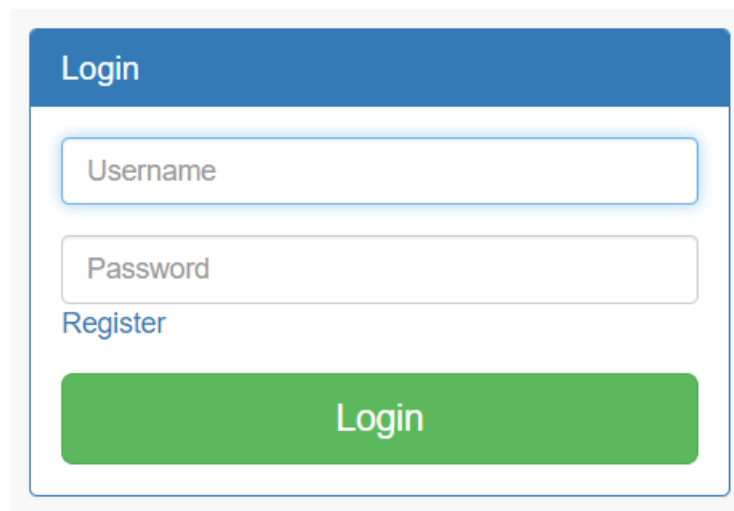
    if(empty(trim($usernamePost))){
        $username_err = "Please enter username.";
    } else{
        $username = trim($usernamePost);
    }

    // Check if password is empty
    if(empty(trim($passwordPost))){
        $password_err = "Please enter your password.";
    } else{
        $password = trim($passwordPost);
    }
}
```

---

*Program code 5.17 Login function*

Picture 11 shows the user interface for login. Under the username and password fields, there is a link to the registration form, if the user does not already have an account.

The image shows a login form with a blue header bar containing the word "Login". Below the header, there are two input fields: "Username" and "Password". Under the "Password" field, there is a blue text link that says "Register". At the bottom of the form, there is a large green button with the word "Login" in white text.

*Picture 12 Login interface*

If the username and password are set, i.e. if the username\_err and password\_err are empty, the variable *\$sql* is filled with a SQL query which searches for the user with the given username. The question mark that is on the end of the SQL query will be replaced with the user's username with the bind\_param() method. Then, the whole query is sent to the database with the execute() method which will then be saved with the store\_result()

method. If the username is in the database, the inputted password will be checked with the password saved in the database.

For security reasons, all passwords are encrypted during registration, and because of that can't be directly compared. For comparing passwords, the *password\_verify()* function is used, which will encrypt the inputted password with the same algorithm as the one in the database, and will then compare them. If the function returns „true“, the passwords match and the user is successfully logged in. If the passwords are not the same, the user is notified that the username or password are incorrect. The rest of the *login* function can be seen in program code 5.18

---

```
if(empty($username_err) && empty($password_err)){
    // Creating SQL query
    $sql= "SELECT email, password FROM users WHERE email= ?";

    if($stmt = $mysqli->prepare($sql)){
        $param_username = $username;

        // Replacing the question mark with the username
        $stmt->bind_param("s", $param_username);

        if($stmt->execute()){
            // Saving result
            $stmt->store_result();

            // check if username is found
            if($stmt->num_rows == 1){

                $stmt->bind_result($username,
$hashed_password);
                if($stmt->fetch()){
                    if(password_verify($password,
$hashed_password)){

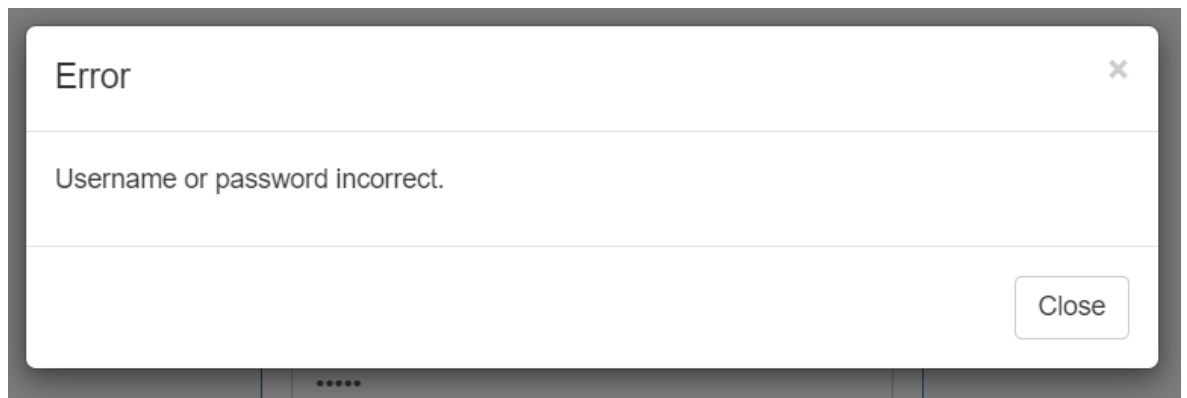
                        return true;
                    } else{

                        $login_err = "Invalid username or
password.";
                    }
                }
            } else{
                $login_err = "Invalid username or
password.";
            }
        }
    }
}
```

---

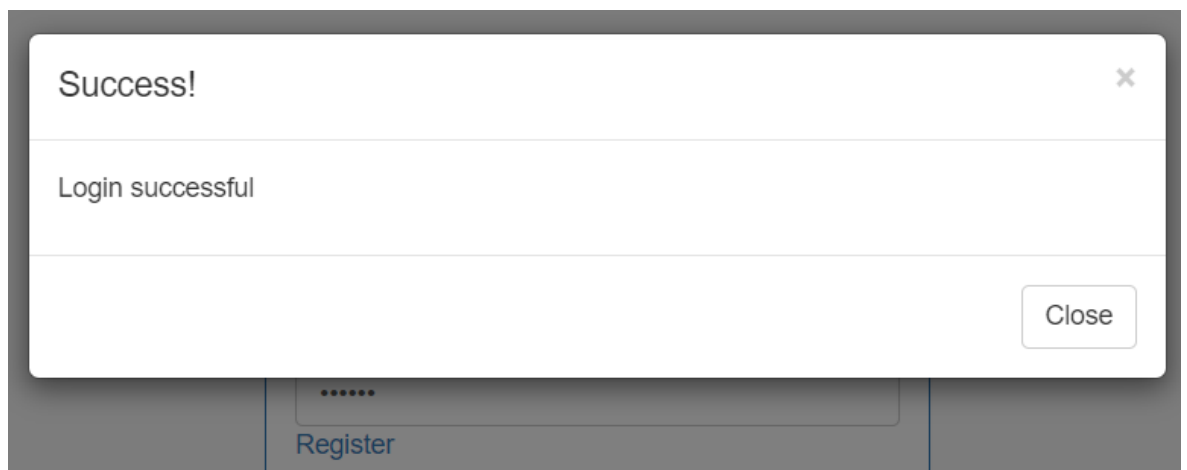
Program code 5.18 Continuation of the login function

Picture 13 shows the notification when the username or password is incorrect, after which the user has the option to login again.



Picture 13 Login unsuccessful

Picture 14 shows the notification when the login is successful, after which the user is redirected to the homepage.



Picture 14 Login successful

#### 2.5.4 Logging into a file

Logging into a file can be used for fixing some program bugs and for database oversight, i.e. having info what the user did at a given moment. In this case, the username and time is saved in the file when the user tried to login.

The `file_put_contents($file,$text,FILE_APPEND)` function is used with the specified parameters. The first parameter is the path to the file in which the data will be written. The second parameter is the text that will be added to the file. The last parameter allows the text to be appended in the file, without it, the previous text would be replaced with the latest text.

In program code 5.19, the first parameter is `log.txt` because the file is already in the same directory in which the login website is. The second parameter takes the `$date` variable and the `$usernamePost` variable. Variable `$date` has the current date and time, while the `$usernamePost` has the user's username. At the end of the second parameter, the „\n“ is added, which is the newline character. It allows us to read the file more easily because every new record will be in its separate line.

---

```
//Logging in a file
File_put_contents('log.txt','On '. $date . ' user '.
$usernamePost . ' logged in'. \n, FILE_APPEND)
```

---

*Program code 5.19 Program code for logging into a file*

On picture 15, an example of writing in the log.txt file can be seen.

```
On 23.04.04 08:23:26 user test2@gmail.com logged in
On 23.04.04 08:26:21 user test2@gmail.com logged in
On 23.04.04 08:27:05 user test2@gmail.com logged in
On 23.04.04 08:28:50 user test2@gmail.com logged in
On 23.04.04 09:09:44 user test1@gmail.com logged in
On 23.04.04 09:24:11 user test1@gmail.com logged in
```

*Picture 15 Logging into a file*

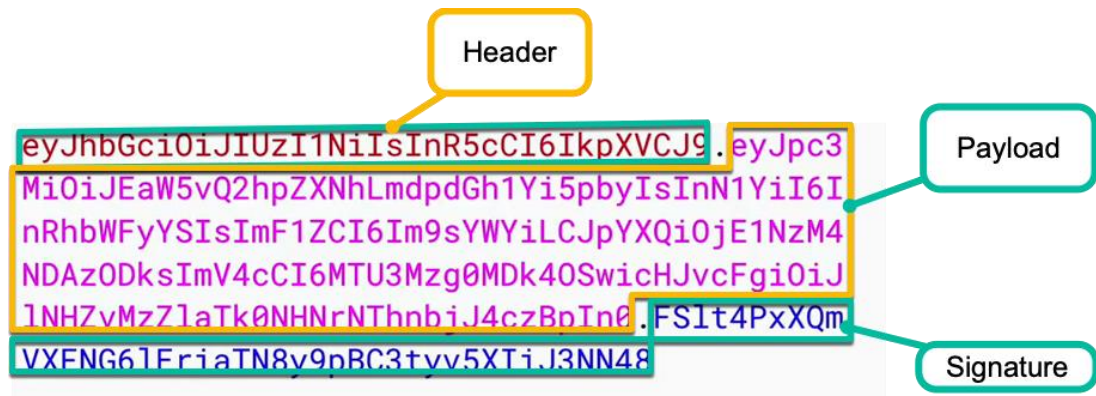
### 2.5.5 Logging in using JWT

In the example above, sessions are used to store data about the user, which could be replaced by JWTs (JSON Web Token). The main reason why JWTs are better to use than sessions is because of security and performance reasons.

The security problem is unauthorized access. During the creation of the session variable, it is saved to a private directory on the server. Although this directory can only be accessed with administrative privileges, there is always the possibility that someone could gain unauthorized access to the directory and collect session data from the users.

Performance problems can occur if several tens of users work on the site at the same time. Since every reading and writing of session data requires a call to the server, there is a possibility of slowing down the server due to an excessive number of requests. Both problems could be solved by using the mentioned tokens.

JWT consists of 3 parts, the header, payload and the signature, which are all separated by a comma. Picture 16 shows an example of an encrypted JWT.



Picture 16 Structure of a JWT

First part of a JWT is the header, which consists of 2 values, the type of the token and the algorithm used to encrypt it. In this case the type of the token is JWT, while the encryption algorithm is HMAC SHA256, abbreviated as HS256. The decrypted header can be seen in program code 5.20

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

Program code 5.20 Decrypted JWT header

The second part is the payload which will have all the data which will be sent. Payload can have custom variable names(private claims), and registered variable names(registered claims). Private claims are used to store data about the user, like his email, privileges etc. Registered claims are variables that are already pre registered and they are used for information about the token. Some of the more important ones are *iss*, *iat*, *nbj*, *exp*. *Iss* will store the issuer info, which will be the website domain. *Iat*(issued at) will have the timestamp of when the token was issued. *Nbj*(not before) will have the timestamp of when the token becomes valid. Its value has to be greater or equal as the *iat* variable. *Exp* (Expiration time) will have the timestamp that indicates when the token expires. It has to be greater than the *iat* and *nbj* variables. In program code 5.21, an example of the JWT payload is shown.

```
{
  "iat": 1677430609,
  "nbj": 1677430609,
  "exp": 1677432000,
  "userEmail": "ehalupecki@vub.hr"
}
```

Program code 5.21 Decrypted payload of a JWT

The last part is the signature, which takes the header and the payload in the base64 format, along with the secret key, and encrypts them using the function chosen in the header. The secret key can be any alphanumeric combination of 256 bits. Program code 5.22 shows an example of creating a signature.

---

```
HMACSHA256(  
  base64UrlEncode(header) + "." +  
  base64UrlEncode(payload) ,  
  secret)
```

---

*Program code 5.22 Creation of the signature of a JWT*

The purpose of the signature is to ensure that the token has not been changed by a third party which could potentially change the data in the header or the payload because they are not encrypted. That is the reason why a signature check is done which ensures that the signature is valid. Picture 17 shows an online debugger which can create a JWT. The newly created JWT used the data from the JWT above including the secret key „tajna“.

Encoded <small>PASTE A TOKEN HERE</small>	Decoded <small>EDIT THE PAYLOAD AND SECRET</small>
<pre>eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpYXQiOiJlY2NzcmZlZm5iZiI6MTY3NzQzMjYwOSwiZXhwIjoxNjc3NDMyMDAwLCJ1c2VyRW1haWwiOiJlaGFsdXB1Y2tpQHZ1Yi5ociJ9.Mp5S2Q09nvSoWw3JB0R_em8sfHd_KuI6LoweZ1l2Zn8</pre>	<div>HEADER: ALGORITHM &amp; TOKEN TYPE</div> <pre>{   "alg": "HS256",   "typ": "JWT" }</pre> <div>PAYLOAD: DATA</div> <pre>{   "iat": 1677430609,   "nbf": 1677430609,   "exp": 1677432000,   "userEmail": "ehalupecki@vub.hr" }</pre> <div>VERIFY SIGNATURE</div> <pre>HMACSHA256(   base64UrlEncode(header) + "." +   base64UrlEncode(payload),   tajna ) <input type="checkbox"/> secret base64 encoded</pre>

*Picture 17 Creation of a JWT with the help of an online debugger*

But if somebody tried to change a part of the *payload*, the *signature* would also change, which would result in an incorrect signature and rejecting the user's login. An example of changing the userEmail field and the new look of the token can be seen on the picture 18.

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpYXQiOiE2Nzc0MzA2MDksIm5iZiI6MTY3NzQzMDEyOTY0IiwiaWF0Ij0zXN0QHRlc3QuaHRlZQ.QzZ1z-Js4P8fWqEXK6sQT9v04sweYb5rfJa\_Ih7qASA

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYLOAD: DATA

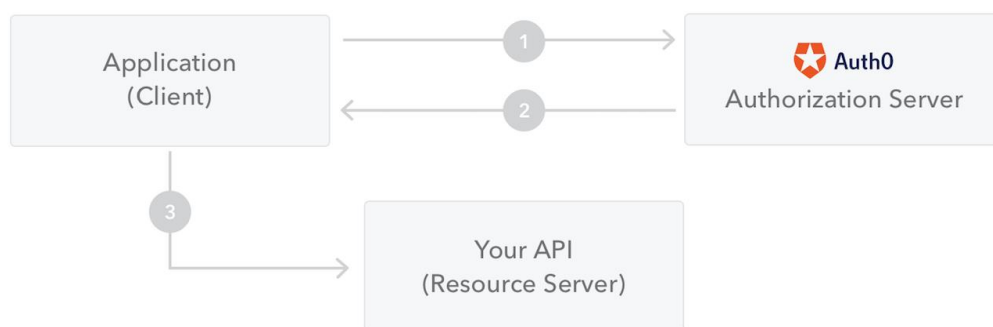
```
{
  "iat": 1677430609,
  "nbf": 1677430609,
  "exp": 1677432000,
  "userEmail": "test@test.hr"
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  tajna
)
```

☐ secret base64 encoded

On picture 19, the authorization workflow using a JWT is shown.



When the user wants to access the website which requires authorization, he will then send the data needed for logging in, and will in return receive a JWT. After that, the user will access the web pages that require authorization and will send the token to the server within the authorization header and the bearer schema. An example of the authorization header can be seen in program code 5.23

Authorization: Bearer

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpYXQiOiJlE2Nzc0MzA2MDksIm5iOiI6MTY3NzQzMdYwOSwiZXhwIjoxNjc3NDMyMDAwLCJ1c2VyRWlhaWwIiOiJlaGFsdXB1Y2tpQHZ1Yi5ociJ9.Mp5S2Q09nvSoWw3JB0R\_em8sfHd\_KuI6LoweZ112Zn8

---

*Program code 5.23 Authorization header*

This way ensures that only the user with the correct JWT can access the website.



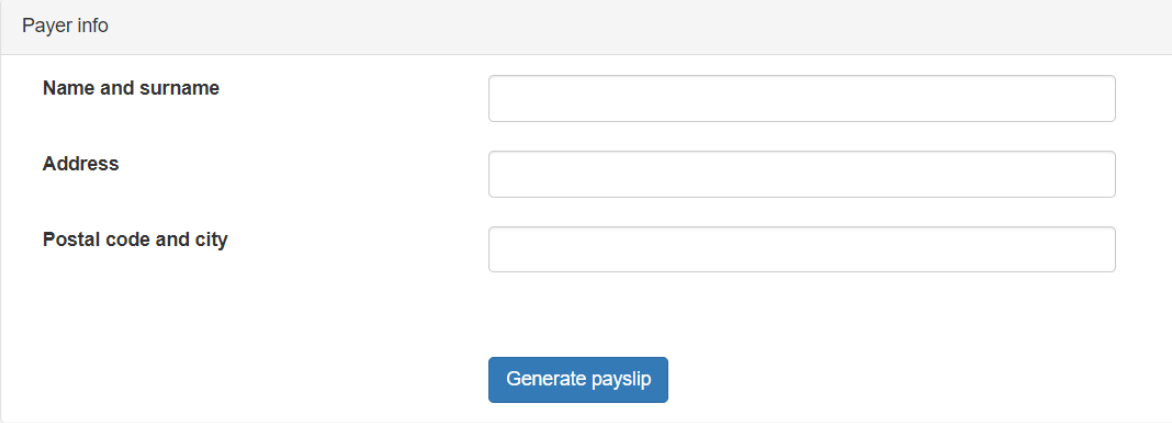
## 2.6 Implement a call to a web service for retrieving an image in PHP

Retrieving the picture is done using the API for generating barcodes for HUB-3 payment slips. In order to generate a valid barcode, additional information is needed, like the name and surname of the recipient, IBAN, amount, model, and other information that is normally used to fill out a HUB-3 payment slip.

### 2.6.1 Call to a web service using the Curl library

*Curl* is an open source library used for communication with servers, i.e. for sending and receiving data. It can be used in most programming languages, and it is most often used for various calls to API (application programming interface) services. In order to retrieve data, it is necessary to define a website from which *curl* will retrieve data. Sending data also requires a website location and parameters that will be sent. Usually, the communication is two-way, that is, in the same call parameters will be sent and a response from the web service will be received.

After clicking the button „Generate pay slip“ on the interface shown on the picture 20, all data is sent to the given URL using Curl. Before sending the data, integer values have to again be converted into integer using the `intval()`. The reason why the conversion is needed is because when the data arrives from frontend in JSON format, everything is converted into string. Finally, the data is then converted into a JSON array with the `json_encode()` function because the API only accepts JSON array as a valid datatype. Interface for generating the pay slip can be seen on picture 20.



The image shows a web form titled "Payer info" with a light gray header. Below the header, there are three input fields stacked vertically. The first field is labeled "Name and surname", the second "Address", and the third "Postal code and city". Each field has a light gray border and a small "x" icon on the right side. Below these fields, there is a blue button with white text that says "Generate payslip".

Picture 20. Interface for generating a pay slip

In program code 6.1, a JSON object is shown that is sent to the API. Option *renderer : image* indicates that an image of the barcode will be sent as a response. The options will have data about how the barcode will look like, i.e. the size, color, format in which the picture will be sent etc. The „*Data*“ consists of parameters that are needed for a transaction, like the name and surname of the sender and receiver, amount and the IBAN. In this example, the data about the receiver is not changed, while the data about the sender is fetched from the form.

---

```

var paymentData= {
  "renderer": "image",
  "options": {
    "format": "png",
    "color": "#000000"
  },
  "data": {
    "amount": 200,
    "sender": {
      "name":
document.getElementById(nameAndSurnameSender).value,
      "street":
document.getElementById('senderAddress').value,
      "place":
document.getElementById('senderCity').value
    },
    "receiver": {
      "name": "Big Fish Software d.o.o.",
      "street": "Savska cesta 13",
      "place": "10000 Zagreb",
      "iban": "HR6623400091110651272",
      "model": "00",
      "reference": "123-456-789"
    },
    "purpose": "ANTS",
    "description": "Developing a HUB-3 API"
  }
};

```

---

*Program code 6.1 JSON object that is sent to the API*

JSON object that was sent to the middle layer is stored in the `$_POST` variable, from which the data can be read. Program code 6.2 shows the process of sending a call to a web service. `$url` variable contains a link to the web service. With the help of the `curl_init()` function, a new session is created for curl which is then forwarded to the `curl_setopt()` function that is used for defining the settings. Firstly, the variable `$url` and session are added to the `CURLOPT_URL` parameter, which sets the destination that Curl will connect to. In this case, `$url` will contain the url to the web service, while the session will be a new one, created with the `curl_init` function

Enabling HTTP POST allows sending data to a web service using the POST method. Without enabling this option, sending the data to the web service via the POST method would not be possible which is required to use this web service. Setting a header is done by sending the `CURLOPT_HEADER` parameter to the `curl_setopt` function along the datatype that is being sent. In this case, a JSON object is sent, so the `array('Content-Type:application/json')` is set.

Last setting that has to be determined before defining which data will be sent is `CURLOPT_RETURNTRANSFER`. If this setting is set to „True“, a response from the web

service that is not 0 or 1 will be possible, i.e. the service will be able to give a detailed response, instead of only sending the information that the data has been received or not.

Because of the conversion that occurs when sending a JSON object from the interface to the middle layer, datatypes that are sent as integers are converted into strings. Because of that, those values have to be reconverted into integers because the web service only accepts integers as datatypes for certain fields. In this case, only the values that define the look and the amount that the user has to pay have to be reconverted. Both of those values will not be changed by the user, and because of that they can be hardcoded.

Last setting that has to be determined is `CURLOPT_POSTFIELD` which is used to determine which data will be sent to the web service. Here, the values collected from the interface will be converted into a string using `json_encode()` function so that the web service can read the values. Lastly, the `$barcode` variable is filled with `curl_exec` function, which will return the barcode image if all the data is valid. Then, the image will be saved on the server and sent to the user's email address.

---

```
$url ="https://hub3.bigfish.software/api/v1/barcode";

$ch = curl_init();

curl_setopt($ch, CURLOPT_URL, $url);

// Enable HTTP POST
curl_setopt($ch, CURLOPT_POST, 1);

//setting header
curl_setopt($ch, CURLOPT_HTTPHEADER, array('Content-
Type:application/json'));

// Enables response different from 0 or 1
    curl_setopt($ch, CURLOPT_RETURNTRANSFER, true);

    $_POST['data']['amount'] = 200;
    $_POST['options']['scale'] = 3;
    $_POST['options']['ratio'] = 3;
    $_POST['options']['padding'] = 20;

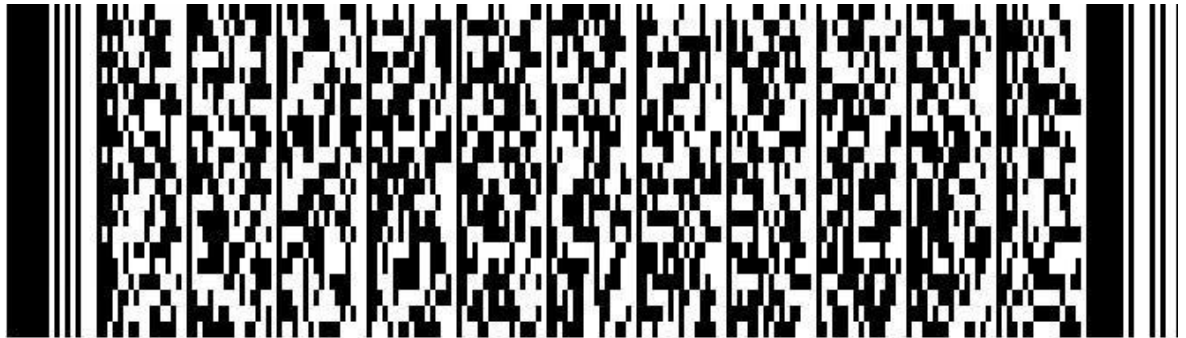
//Setting up parameters that are going to be sent
curl_setopt($ch, CURLOPT_POSTFIELDS, json_encode($_POST));

$barcode = curl_exec($ch);
```

---

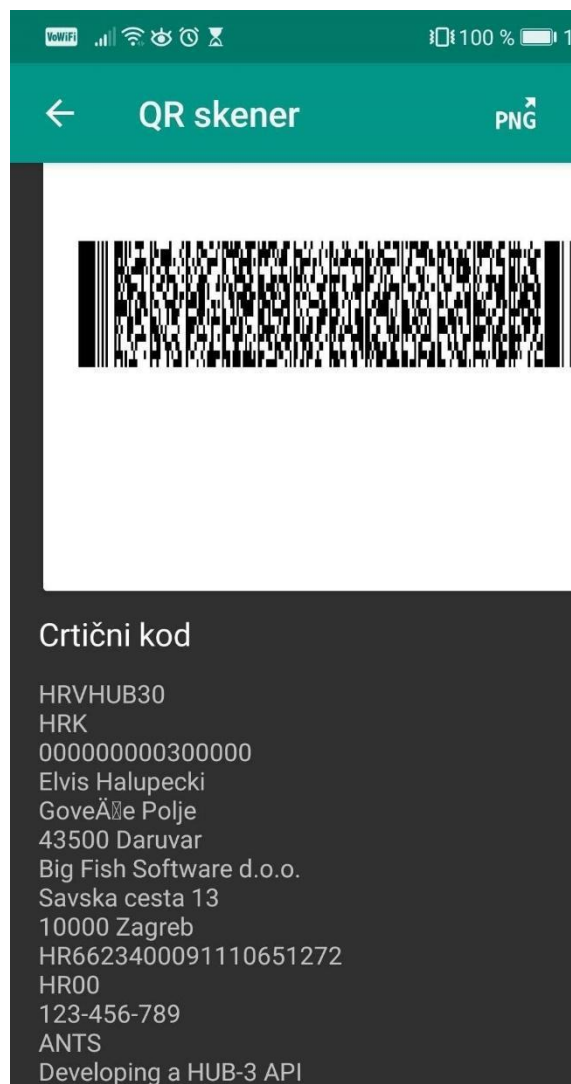
*Program code 6.2 Call towards the API*

If the data is in the correct format, the API returns a picture of the barcode with the entered data, which can then be scanned and the payment slip can be paid.



*Picture 21 Example of the barcode returned by the web service*

Scanning the barcode shown on picture 21 with a mobile phone, returns the information about the payee and the payer, which is shown on picture 22.



*Picture 22. Barcode scanned on a mobile phone*

### **3. CONCLUSION**

Using the Internet and browsing websites has become a daily routine in the modern world. For this reason, more and more investments are being made in the development of technologies specifically aimed at web environments. Of security due to new loopholes that are found and new ways to bypass current protections. For this reason, login using JWT is shown, which offers better security, and the use of prepared statements has also been shown, which reduces the chance of SQL injection. The role of the middle layer will always be the first „line of defence“ to ensure that only the valid data reaches the database.

In addition to the security functions, the thesis consists of some other functions that can be easily implemented using the middle layer. Creating word documents and parameterizing fields depending on the user is simple, so the only action left for creating official documents is to add an official signature. Reports and statistics can also be created on the middle layer. Those reports can be simple, consisting only of a table and records, or they can be more complex, by adding graphs, diagrams and other similar visual representations of data which can then be sent to specific email addresses, as shown in the thesis itself.

There is always room for improvement, from better encryption of the passwords in the database, to improving the code which would eliminate the threats of denials of services and other advanced attack methods. But for this project those improvements were not necessary because the expected results were obtained and because the project itself was used only as a proof of concept.

## 4. LITERATURE

- [1] <https://www.freecodecamp.org/news/what-is-middleware-with-example-use-cases/>, FreeCodeCamp.com
- [2] <https://www.php.net/manual/en/session.examples.basic.php>, Php.net
- [2] [https://www.w3schools.com/php/php\\_sessions.asp](https://www.w3schools.com/php/php_sessions.asp), w3schools.com
- [4] <https://www.php.net/manual/en/function.header.php>, php.net
- [5] [https://www.w3schools.com/php/func\\_network\\_header.asp](https://www.w3schools.com/php/func_network_header.asp), w3schools.com
- [6] <https://phpword.readthedocs.io/en/latest/>, phpword.readthedocs.io
- [7] <https://phpspreadsheet.readthedocs.io/en/latest/>, phpspreadsheet.readthdocs.io
- [8] <https://www.cloudways.com/blog/send-emails-in-php-using-phpmailer/>, cloudways
- [2] <https://stackoverflow.com/questions/28010440/how-to-send-data-in-json-format-in-jquery-ajax-for-restfull-web-services>, stackoverflow.com
- [10] <https://www.phptutorial.net/php-oop/>, phptutorial.net
- [11] <https://www.devopsschool.com/blog/complete-tutorials-of-php-oop-constructor-with-example-code/>, devopsschool.com
- [12] <https://www.php.net/manual/en/book.pdo.php>, php.net
- [13] <https://www.guru99.com/mysql-php-and-other-database-access-methods.html>, guru99.com
- [14] <https://www.php.net/manual/en/book.mysqli.php>, php.net
- [15] <https://www.phpclasses.org/blog/post/521-mysqli-vs-pdo-vs-mysql.html>, phpclasses.org
- [16] <https://stackoverflow.com/questions/5592215/why-use-an-orm-in-php>, stackowerflow.com
- [17] <https://www.doctrine-project.org/projects/doctrine-orm/en/2.14/index.html>, doctrine-project.org
- [18] <https://www.tutorialrepublic.com/php-tutorial/php-mysql-login-system.php>, tutorialrepublic.com
- [19] <https://jwt.io/introduction>, jwt.io
- [20] <https://medium.com/swlh/why-do-we-need-the-json-web-token-jwt-in-the-modern-web-8490a7284482>, medium.com
- [21] <https://auth0.com/docs/secure/tokens/json-web-tokens>, auth0.com
- [22] <https://hub3.bigfish.software/api/v1>, hub3.bigfish.software

## **5. ABBREVIATIONS AND ACRONYMS**

API – Application Programming Interface

JSON – JavaScript Object Notation

JWT – JSON Web Token

ORM – Object-Relational Mapping

PDO – PHP Data Objects

XML – Extensible Markup Language

## 6. ABSTRACT

**Title:** The role of middle layer in web environment

The final thesis explains the purpose and importance of the middle layer in a web environment. It also shows the usage of session and headers to achieve better user experience. Generating templates is achieved by using the PHPOffice library. It shows the process of creating a template and the final result. It also shows the code code needed to send emails from the middle layer and ways to send data from frontend to the middle layer with the help of AJAX calls. The login process with a username and password are also shown, including code which validates data that comes from the user during logging in. There is also a comparison between PDO and MySQLi extensions which are used to communicate with the database. It is explained what PHP ORM library is, and the reasons why it's used. There is also a comparison between standard login and login with JWT, including the advantages of JWTs. cURL functions are used to call web services which generate barcodes that are used to pay HUB-3 payment slips.

**Keywords:** session, header, AJAX calls, PDO, MySQLi, PHP ORM, JWT, cURL functions



## 7. SAŽETAK

**Naslov:** Uloga srednjeg sloja u web okruženju

U radu se opisuje uloga i važnost srednjeg sloja u web okruženju. Prikazuje se korištenje sesija i zaglavlja kako bi se ostvarilo bolje korisničko iskustvo. Generiranje predložaka je objedinjeno u PHPOffice biblioteci te je prikazan proces koji je potreban za generiranje predložaka te finalni rezultat, odnosno generirana datoteka. Prikazan je programski kod potreban za slanje e-mailova s srednjeg sloja, te način na koji se podaci šalju sa sučelja na srednji sloj uz pomoć AJAX poziva. Uz sami prikaz slanja podataka, prikazan je proces prijave s korisničkim imenom i zaporkom te programski kod pomoću kojeg se provjerava točnost podataka koje korisnik upisuje prilikom prijave. Prikazana je usporedba PDO i MySQLi ekstenzija koje služe za korištenje baza podataka. Prikazana je namjena PHP ORM biblioteka te razlozi zbog kojih bi mogle biti poželjne za korištenje. Navedeni su razlozi zbog kojih bi prijava pomoću JWT-a bila bolja od uobičajene prijave. Uz pomoć cURL funkcija se poziva web servis koji prima parametre pomoću kojih se generira barkod koji se vraća korisniku pomoću kojeg se može platiti HUB-3 uplatnica.

**Ključne riječi:** sesija, zaglavlja, AJAX poziv, PDO, MySQLi, PHP ORM, JWT, cURL funkcija.

## IZJAVA O AUTORSTVU ZAVRŠNOG RADA

Pod punom odgovornošću izjavljujem da sam ovaj rad izradio/la samostalno, poštujući načela akademske čestitosti, pravila struke te pravila i norme standardnog hrvatskog jezika. Rad je moje autorsko djelo i svi su preuzeti citati i parafraze u njemu primjereno označeni.

Mjesto i datum	Ime i prezime studenta/ice	Potpis studenta/ice
U Bjelovaru, <u>13. 1. 2023</u>	ELVIS HALUPECKI	<i>Edin Elvis</i>

U skladu s čl. 58, st. 5 Zakona o visokom obrazovanju i znanstvenoj djelatnosti, Veleučilište u Bjelovaru dužno je u roku od 30 dana od dana obrane završnog rada objaviti elektroničke inačice završnih radova studenata Veleučilišta u Bjelovaru u nacionalnom repozitoriju.

Suglasnost za pravo pristupa elektroničkoj inačici završnog rada u nacionalnom repozitoriju

ELVIS HALUPECKI

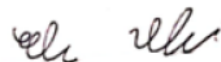
*ime i prezime studenta/ice*

Dajem suglasnost da tekst mojeg završnog rada u repozitorij Nacionalne i sveučilišne knjižnice u Zagrebu bude pohranjen s pravom pristupa (zaokružiti jedno od ponuđenog):

- ☒ a) Rad javno dostupan  
b) Rad javno dostupan nakon \_\_\_\_\_ (upisati datum)  
c) Rad dostupan svim korisnicima iz sustava znanosti i visokog obrazovanja RH  
d) Rad dostupan samo korisnicima matične ustanove (Veleučilište u Bjelovaru)  
e) Rad nije dostupan

Svojim potpisom potvrđujem istovjetnost tiskane i elektroničke inačice završnog rada.

U Bjelovaru, 23.1.2023



*potpis studenta/ice*