REST API Authentication Example in PHP – JWT Tutorial

Last Update: September 22, 2019 • Date Posted: September 19, 2018 • by Mike Dalisay ➜ Get FREE Updates Here

[https://i0.wp.com/www.codeofaninja.com/wp-content/uploads/2018/09/rest-api-authentication-example-banner-2.jpg?ssl=1]

Previously, we learned how to create a simple REST API in PHP [https://www.codeofaninja.com/2017/02/create-simple-rest-api-in-php.html] . The create, read, update and delete database records (CRUD operations) has been useful for our projects.

Today, we will learn how to authenticate a user using REST API and JSON Web Tokens or JWT.

In this tutorial, we will cover a basic sign up or registration form, login and logout operations, updating a user account and more.

1.0 Project overview
1.1 What is JWT?
1.2 JWT simple analogy
1.3 What does a JWT look like?
1.4 JWT vs OAuth

2.0 Final output
3.0 File structure

4.0 Setup the database
4.1 Create a database
4.2 Create a table
4.3 Create a directory for configuration
4.4 Create a database connection file

5.0 Create API for user registration
5.1 Create a file for creating a user
5.2 Connect to database and user table
5.3 Assign submitted data to object properties
5.4 Use the create() method
5.5 Create the user object class
5.6 Add a create() method
5.7 Output

6.0 Create API for user login
6.1 Create a file for user login
6.2 Connect to database and user table
6.3 Check if email exists
6.4 Add emailExists() method
6.5 Include files to encode JWT
6.6 Generate JSON web token
6.7 Tell the user login failed
6.8 Create core configuration file
6.9 Download PHP-JWT from GitHub
6.10 Output

7.0 Create API for JWT validation
7.1 Create a file to validate JWT

7.2 Include files to decode JWT
7.3 Retrieve given JWT
7.4 Decode JWT if it exists
7.5 Show error if decoding failed
7.6 Show error message if JWT is empty
7.7 Output

8.0 Create API for user account
8.1 Create a file for updating user account
8.2 Include files to decode JWT
8.3 Connect to database and user table
8.4 Retrieve given JWT
8.5 Decode JWT if it exists
8.6 Show error message if decoding fails
8.7 Set user property values
8.8 Use the update() method
8.9 Add update() method in user class
8.10 Re-generate JWT
8.11 Show error message if JWT is empty
8.12 Output

9.0 Create interface for user registration
9.1 Create index page
9.2 Add navigation bar
9.3 Add content section
9.4 Add Bootstrap 4 and custom CSS links
9.5 Create custom CSS file
9.6 Add jQuery and Bootstrap 4 script links
9.7 Show a sign up HTML form
9.8 Trigger when sign up form is submitted
9.9 Remove any prompt messages
9.10 Add serializeObject() function
9.11 Output

10.0 Create interface for user login
10.1 Trigger when login menu was clicked
10.2 Show login HTML form
10.3 Add setCookie() function
10.4 Change menu appearance
10.5 Trigger when login form is submitted
10.6 Create an HTTP request

1.0 PROJECT OVERVIEW

1.1 What is JWT?

In technical terms, JSON Web Token or JWT is an open standard (RFC 7519 [https://tools.ietf.org/html/rfc7519]) that defines a compact and self-contained way for

securely transmitting information between parties as a JSON object.

This information can be verified and trusted because it is digitally signed. JWTs can be signed using a secret (with the HMAC algorithm) or a public/private key pair using RSA or ECDSA.

For example, a server could generate a token that has the claim "logged in as admin" and provide that to a client. The client could then use that token to prove that it is logged in as admin. Please read more here [https://jwt.io/introduction/] , here [https://en.wikipedia.org/wiki/JSON_Web_Token] , here [https://medium.com/vandium-software/5-easy-steps-to-understanding-json-web-tokens-jwt-1164c0adfcec] and here [https://medium.com/ag-grid/a-plain-english-introduction-to-json-web-tokens-jwt-what-it-is-and-what-it-isnt-8076ca679843] .

1.2 JWT simple analogy

The following is my own simple analogy. If you think you have a better one, please let me know via email. My email address is `mike@codeofaninja.com`

John (server) owns a house (protected data). Michael (client) wants to rent this house. John and Michael agreed to the house rules and they signed a contract (valid username and password).

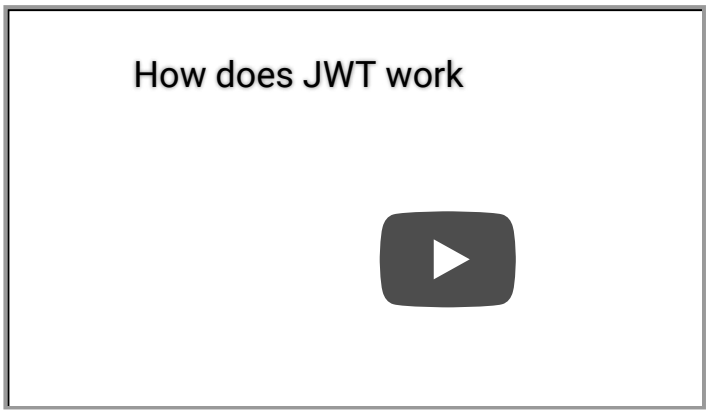
John gave Michael a key (token) with other related information (claims) so he has access to the house. Michael can now get inside or outside (HTTP requests) the house. It means an access was granted.

If John (sever) and Michael (client) did not agree with the house rules, the contract won't be signed (invalid username and password) and John (server) won't give him a key (token).

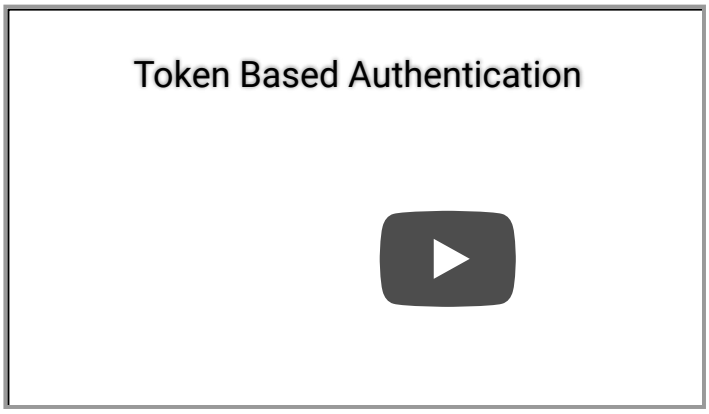
Even if Michael (client) has another type of key (token), he still won't have access to the house because it is a wrong key (token). It is not the key (token) John gave. It means access was denied.

I've found another analogy that can be useful for you. Read it here [https://dev.to/hemanth/explain-jwt-like-im-five] .

The video below might help explain the analogy.



The following video about token-based authentication might help as well.



1.3 What does a JWT look like?

A JSON Web Token or JWT looks like a string with three parts separated by dots. The

following is an example of JWT.

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.ey
```

JWT in the serialized form represents a string of the following format:

```
header.payload.signature
```

The `header` component contains information about how JWT signature should be computed. The `payload` component is the data that is stored inside the JWT. This can be user information like user ID, name and email.

To create the `signature` component, you have to take the encoded header, the encoded payload, a secret, the algorithm specified in the header, and sign that. Read more here [https://jwt.io/introduction/] .

In this tutorial, we won't have to worry about generating or encoding and decoding JWT because we will use a library called PHP-JWT [https://github.com/firebase/php-jwt] .

1.4 JWT vs OAuth

We explained JWT above. JWT is a token format and we can say it is a simple authentication protocol. OAuth [https://en.wikipedia.org/wiki/OAuth] is an authentication framework that can use JWT as a token.

OAuth [https://oauth.net/] is used as a way for Internet users to grant websites or applications access to their information on other websites but without giving them the passwords.

Use JWT if:

- You have very simple use-case, like a single client application.
- Your users access their resources only through your own application.
- You want a quick-to-implement and simple stateless HTTP authentication to an API.

Use OAuth if:

- Your users can access their resources through another application you don't own.
- You want to provide API to browser-based apps, native mobile apps or desktop apps.
- You want to use an Authentication Server that keeps track of tokens.

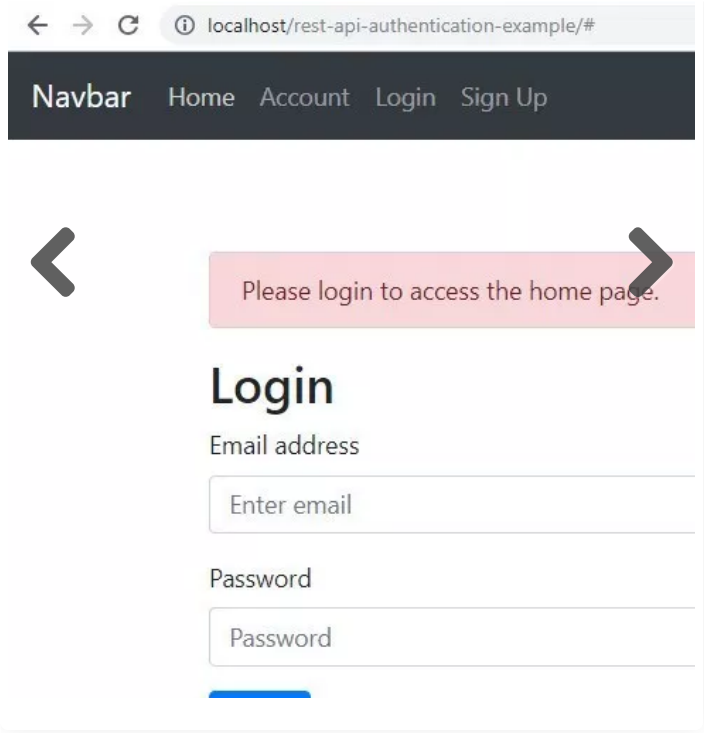
Please read more here [https://stackoverflow.com/questions/39909419/jwt vs-oauth-authentication] and here [https://community.apigee.com/questions/21139/jwt vs-oauth.html] .

2.0 FINAL OUTPUT

2.1 LEVEL 1 source code output

It is important to visualize what we are trying to achieve in this tutorial. At the end of this tutorial, we will achieve the LEVEL 1 source code as seen on the screenshots below.

Please click a photo below to enlarge and use the arrow icons to navigate the slideshow.



2.2 LEVEL 2 source code output

Screenshots coming soon! See the list of features on section 14.2 below.

The LEVEL 2 source code shows more amazing features that you can learn once you completed studying the LEVEL 1 source code.

For now, let's proceed to the complete tutorial of our LEVEL 1 source code below. Let's code!

3.0 FILE STRUCTURE

At the end of this tutorial, we will have the following folders and files.

```
├─ rest-api-authentication-example/ - name of the project folder.
├──── api/ - main folder of the API.
├────────── config/
├──────────────── core.php - file used for common settings or variables.
├──────────────── database.php - file used for
```

connecting to the database.
├──────── libs/
├──────────── php-jwt-master/ - folder of jwt library developed by Google.
├──────── objects/
├──────────── user.php - class file that will handle the database queries.
├──────── create_user.php - file that will process the input of from "sign up" form.
├──────── login.php - file that will encode and generate a JSON web token.
├──────── update_user.php - file that will process the input of from "user account" form.
├──────── validate_token.php - file that will validate or decode the JSON web token.
├──── custom.css - contains any customization in the user interface.
├──── index.html - contains HTML and JavaScript that renders different user interfaces.

4.0 SETUP THE DATABASE

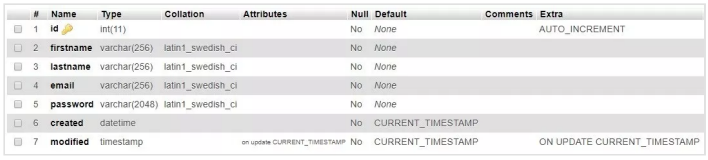
4.1 Create a database

Open your PhpMyAdmin localhost/phpmyadmin [http://localhost/phpmyadmin/] and create a database called `api_db`

4.2 Create a table

On the `api_db` database, create a new table called `users`.

Put the following fields on the `users` table.



#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	id	int(11)			No	None		AUTO_INCREMENT
2	firstname	varchar(256)	latin1_swedish_ci		No	None		
3	lastname	varchar(256)	latin1_swedish_ci		No	None		
4	email	varchar(256)	latin1_swedish_ci		No	None		
5	password	varchar(2048)	latin1_swedish_ci		No	None		
6	created	datetime			No	CURRENT_TIMESTAMP		
7	modified	timestamp		on update CURRENT_TIMESTAMP	No	CURRENT_TIMESTAMP		ON UPDATE CURRENT_TIMESTAMP

4.3 Create a directory for configuration

Create our main project folder and put `rest-api-authentication-example` as its name.

If you're using XAMPP, you must create it inside the `htdocs` folder. In my case, I created it inside `C:\xampp\htdocs` directory.

Open `rest-api-authentication-example` folder. Create `api` folder.

Open `api` folder. Create `config` folder.

4.4 Create a database connection file

Open `config` folder. Create a new file called `database.php`.

Place the following code.

```
<?php
// used to get mysql database connectic
class Database{

    // specify your own database creder
    private $host = "localhost";
    private $db_name = "api_db";
    private $username = "root";
    private $password = "";
    public $conn;

    // get the database connection
    public function getConnection(){

        $this->conn = null;

        try{
            $this->conn = new PDO("mysc
        }catch(PDOException $exception)
            echo "Connection error: " .
        }

        return $this->conn;
    }
}
```

```
?>
```

5.0 CREATE API FOR USER REGISTRATION

5.1 Create a file for creating a user

Open `rest-api-authentication-example` folder. Create a folder called `api`.

Open the `api` folder. Create a new file called `create_user.php`.

We need to set headers on this new file so that it will only accept JSON data. Place the following code.

```
<?php
// required headers
header("Access-Control-Allow-Origin: ht
header("Content-Type: application/json;
header("Access-Control-Allow-Methods: P
header("Access-Control-Max-Age: 3600");
header("Access-Control-Allow-Headers: C

// database connection will be here
```

5.2 Connect to database and user table

We are saving the user information on a database so we need the database connection. We need to instantiate the `user` table as well because this will make the `insert` query later.

Replace `// database connection will be here` comment of `create_user.php` file with the following code.

```
// files needed to connect to database
include_once 'config/database.php';
include_once 'objects/user.php';

// get database connection
```

```
$database = new Database();
$db = $database->getConnection();

// instantiate product object
$user = new User($db);

// submitted data will be here
```

5.3 Assign submitted data to object properties

The user information will be submitted through an HTML form and JavaScript code. We will see this code later.

We need to assign the submitted data on the object properties such as `firstname`, `lastname`, etc.

Replace `// submitted data will be here` comment of `create_user.php` file with the following code.

```
// get posted data
$data = json_decode(file_get_contents('

// set product property values
$user->firstname = $data->firstname;
$user->lastname = $data->lastname;
$user->email = $data->email;
$user->password = $data->password;

// use the create() method here
```

5.4 Use the create() method

One the code below, we use the `user` object's `create()` method. It will tell the user if the user was created or not.

Replace `// use the create() method here` comment of `create_user.php` file with the following code.

```
// create the user
if(
    !empty($user->firstname) &&
    !empty($user->email) &&
    !empty($user->password) &&
```

```
    $user->create()
){

    // set response code
    http_response_code(200);

    // display message: user was create
    echo json_encode(array("message" =>
}

// message if unable to create user
else{

    // set response code
    http_response_code(400);

    // display message: unable to creat
    echo json_encode(array("message" =>
}
?>
```

5.5 Create the user object class

The previous section will not work without the user object class. This is where we'll place all the `user` methods that contains database queries.

If you're not familiar with private or public scopes, please learn from this resource [https://stackoverflow.com/a/21902271] .

Open the `api` folder. Open `objects` folder.

Create a new file called `user.php` . Place the following code.

```
<?php
// 'user' object
class User{

    // database connection and table na
    private $conn;
    private $table_name = "users";

    // object properties
    public $id;
    public $firstname;
    public $lastname;
    public $email;
    public $password;

    // constructor
```

```
    public function __construct($db){
        $this->conn = $db;
    }

// create() method will be here
}
```

5.6 Add a create() method

The code below shows the `INSERT` query, data sanitation and binding, and we used the built-in `password_hash()` method to secure the user's password on the database.

If the execution is a success, the user information will be saved on the database.

Replace the `// create() method will be here` comment of `user.php` file with the following code.

```
// create new user record
function create(){

    // insert query
    $query = "INSERT INTO " . $this->ta
            SET
                firstname = :firstname,
                lastname = :lastname,
                email = :email,
                password = :password";

    // prepare the query
    $stmt = $this->conn->prepare($query

    // sanitize
    $this->firstname=htmlspecialchars(s
    $this->lastname=htmlspecialchars(st
    $this->email=htmlspecialchars(strip
    $this->password=htmlspecialchars(st

    // bind the values
    $stmt->bindParam(':firstname', $thi
    $stmt->bindParam(':lastname', $this
    $stmt->bindParam(':email', $this->e

    // hash the password before saving
    $password_hash = password_hash($thi
    $stmt->bindParam(':password', $pass

    // execute the query, also check if
    if($stmt->execute()){
        return true;
    }
```

```
    return false;
}

// emailExists() method will be here
```

5.7 Output

You need to use POSTMAN [https://www.getpostman.com/] to test our API. Download your version of POSTMAN here [https://www.getpostman.com/apps].

First, we will **test for successful creation of a user**. Launch POSTMAN.

Enter the following as the request URL

```
http://localhost/rest-api-authentication-example/api/create_user.php
```

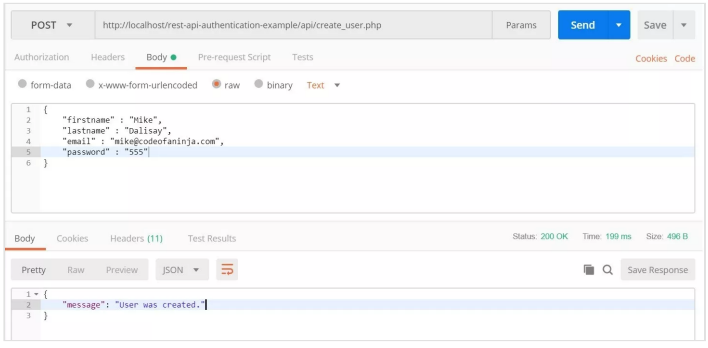
Click "Body" tab. Click "raw". Enter this JSON value:

```
{
    "firstname" : "Mike",
    "lastname" : "Dalisay",
    "email" : "mike@codeofaninja.com",
    "password" : "555"
}
```

Click the blue "Send" button. Output will be:

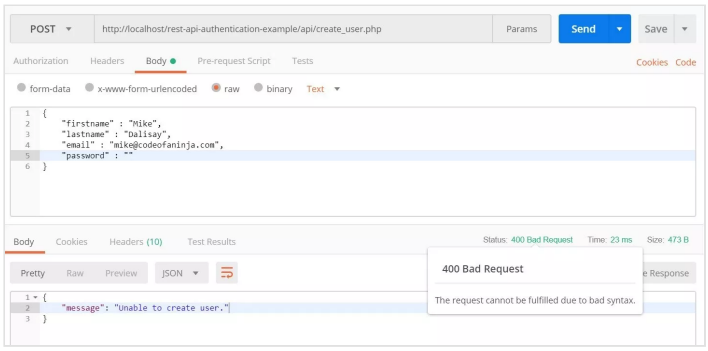
```
{
    "message": "User was created."
}
```

On POSTMAN, it should look like this:



To **test for a failed creation of a user**, just remove the value of the password above and click the blue "Send" button.

It should look like this:



6.0 CREATE API FOR USER LOGIN

6.1 Create a file for user login

On the code below, we set the file headers so that it will know where the request should come from and what type of data is accepted.

Open `rest-api-authentication-example` folder.

Open Create a new file called `login.php`. Place the following code.

```
<?php
// required headers
header("Access-Control-Allow-Origin: ht
header("Content-Type: application/json;
header("Access-Control-Allow-Methods: P
header("Access-Control-Max-Age: 3600");
header("Access-Control-Allow-Headers: C

// database connection will be here
```

6.2 Connect to database and user table

We will compare the user email and password from the database so we need the database

connection.

We need to instantiate the user table as well because this will allow us to verify if the email exists and read the hashed password.

Replace `// database connection will be here` comment of `login.php` file with the following code.

```
// files needed to connect to database
include_once 'config/database.php';
include_once 'objects/user.php';

// get database connection
$database = new Database();
$db = $database->getConnection();

// instantiate user object
$user = new User($db);

// check email existence here
```

6.3 Check if email exists

On the code below, we get the email submitted by the user through the login form. We check if the email exists on our database.

Replace `// check email existence here` comment of `login.php` file with the following code.

```
// get posted data
$data = json_decode(file_get_contents('

// set product property values
$user->email = $data->email;
$email_exists = $user->emailExists();

// files for jwt will be here
```

6.4 Add emailExists() method

We will add an emailExists() method on our `user` object class. This method will return `true` if the submitted email exists, else it will return `false`.

Replace `// emailExists() method will be here` comment of `/api/objects/user.php` file with the following code.

```
// check if given email exist in the da
function emailExists(){

    // query to check if email exists
    $query = "SELECT id, firstname, las
            FROM " . $this->table_name
            WHERE email = ?
            LIMIT 0,1";

    // prepare the query
    $stmt = $this->conn->prepare( $quer

    // sanitize
    $this->email=htmlspecialchars(strip

    // bind given email value
    $stmt->bindParam(1, $this->email);

    // execute the query
    $stmt->execute();

    // get number of rows
    $num = $stmt->rowCount();

    // if email exists, assign values t
    if($num>0){

        // get record details / values
        $row = $stmt->fetch(PDO::FETCH_

        // assign values to object prop
        $this->id = $row['id'];
        $this->firstname = $row['firstr
        $this->lastname = $row['lastnam
        $this->password = $row['passwor

        // return true because email ex
        return true;
    }

    // return false if email does not e
    return false;
}

// update() method will be here
```

6.5 Include files to encode JWT

The code below shows the necessary files we needed to include to generate or encode a

JSON web token.

Replace `// files for jwt will be here` comment of `login.php` file with the following code.

```
// generate json web token
include_once 'config/core.php';
include_once 'libs/php-jwt-master/src/B
include_once 'libs/php-jwt-master/src/B
include_once 'libs/php-jwt-master/src/S
include_once 'libs/php-jwt-master/src/J
use \Firebase\JWT\JWT;

// generate jwt will be here
```

6.6 Generate JSON web token

The code below will check if email exists and if password match what is in the database. We used the built-in `password_verify()` function to do the matching.

If login is valid, it will generate the JSON Web Token.

Replace `// generate jwt will be here` comment of `login.php` file with the following code.

```
// check if email exists and if passwor
if($email_exists && password_verify($da

    $token = array(
       "iss" => $iss,
       "aud" => $aud,
       "iat" => $iat,
       "nbf" => $nbf,
       "data" => array(
           "id" => $user->id,
           "firstname" => $user->firstr
           "lastname" => $user->lastnam
           "email" => $user->email
       )
    );

    // set response code
    http_response_code(200);

    // generate jwt
    $jwt = JWT::encode($token, $key);
    echo json_encode(
```

```
            array(
                "message" => "Successfu
                "jwt" => $jwt
            )
        );

}

// login failed will be here
```

6.7 Tell the user login failed

If the email does not exist or the password did not match, tell the user he cannot login.

Replace `// login failed will be here` comment of `login.php` file with the following code.

```
// login failed
else{

    // set response code
    http_response_code(401);

    // tell the user login failed
    echo json_encode(array("message" =>
}
?>
```

6.8 Create core configuration file

The `login.php` file will not work without the `core.php` file. This file contains common settings or variables of our application.

We have variables used by our JWT library to encode and decode a token. `$key`'s value must be your own and unique secret key.

The rest is called the **registered claim names**. The `iss` (issuer) claim identifies the principal that issued the JWT.

The `aud` (audience) claim identifies the recipients that the JWT is intended for. The

`iat` (issued at) claim identifies the time at which the JWT was issued.

The `nbf` (not before) claim identifies the time before which the JWT MUST NOT be accepted for processing.

You can use another useful claim name called `exp` (expiration time) which identifies the expiration time on or after which the JWT MUST NOT be accepted for processing.

Including these claims are optional. Please read more about registered claim names here [https://tools.ietf.org/html/rfc7519#section-4.1] .

Open the `api` folder. Open the `config` folder. Create a new file called `core.php` . Place the following code.

```
<?php
// show error reporting
error_reporting(E_ALL);

// set your default time-zone
date_default_timezone_set('Asia/Manila'

// variables used for jwt
$key = "example_key";
$iss = "http://example.org [http://exam
$aud = "http://example.com [http://exam
$iat = 1356999524;
$nbf = 1357000000;
?>
```

6.9 Download PHP-JWT from GitHub

The files included in `login.php` file will not work without this library.

Download the library from this link [https://github.com/firebase/php-jwt] .

Create `libs` folder. Unzip the downloaded library there. See the file structure above to see how it should look like.

6.10 Output

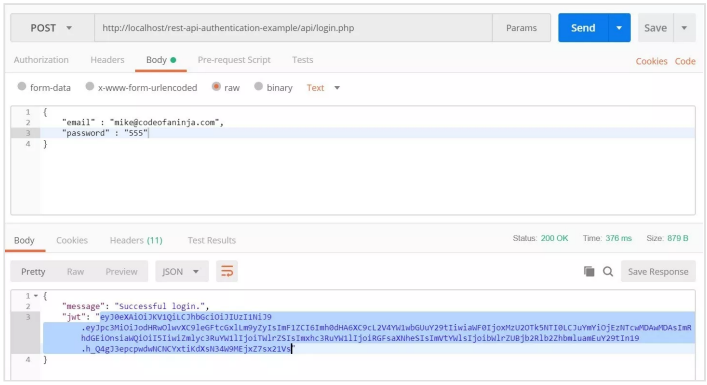
To **test for successful login**, enter the following as the request URL.

```
http://localhost/rest-api-authentication-example/api/login.php
```

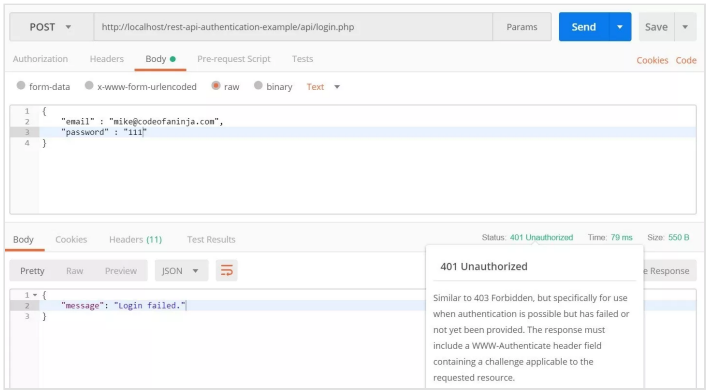
Enter the following on the body.

```
{
    "email" : "mike@codeofaninja.com",
    "password" : "555"
}
```

We need to take note of the generated JWT because we will use it to access a resource later.



To **test for failed login**, change the value of the password to `111` because it is a wrong password.



7.0 CREATE API FOR JWT VALIDATION

7.1 Create a file to validate JWT

This file will return an output in JSON format and will accept requests from specified URL. We'll set the correct headers.

Open `api` folder. Create `validate_token.php` file. Place the following code.

```
<?php
// required headers
header("Access-Control-Allow-Origin: ht
header("Content-Type: application/json;
header("Access-Control-Allow-Methods: P
header("Access-Control-Max-Age: 3600");
header("Access-Control-Allow-Headers: C

// files for decoding jwt will be here
```

7.2 Include files to decode JWT

The code below shows the inclusion of the necessary files to decode the given JSON web token.

Replace `// files for decoding jwt will be here` comment of `validate_token.php` file with the following code.

```
// required to decode jwt
include_once 'config/core.php';
include_once 'libs/php-jwt-master/src/B
include_once 'libs/php-jwt-master/src/B
include_once 'libs/php-jwt-master/src/S
include_once 'libs/php-jwt-master/src/J
use \Firebase\JWT\JWT;

// retrieve gieve jwt here
```

7.3 Retrieve given JWT

The code below shows how to get the value of JSON web token.

Replace `// retrieve gieve jwt here` comment of `validate_token.php` file with the following code.

```
// get posted data
$data = json_decode(file_get_contents('

// get jwt
$jwt=isset($data->jwt) ? $data->jwt : '

// decode jwt here
```

7.4 Decode JWT if it exists

Check if a JWT is given. If true, decode it. Return a response code of 200, tell the user access is granted and some user information.

Replace `// decode jwt here` comment of `validate_token.php` file with the following code.

```
// if jwt is not empty
if($jwt){

    // if decode succeed, show user det
    try {
        // decode jwt
        $decoded = JWT::decode($jwt, $k

        // set response code
        http_response_code(200);

        // show user details
        echo json_encode(array(
            "message" => "Access grante
            "data" => $decoded->data
        ));

    }

    // catch will be here
}

// error if jwt is empty will be here
```

7.5 Show error if decoding failed

If decoding JWT failed, it means the access to the resource is denied. We need to return a response code of 401, tell the user access is denied and some information about the error.

Replace `// catch will be here` comment of `validate_token.php` file with the following code.

```
// if decode fails, it means jwt is inv
catch (Exception $e){

    // set response code
    http_response_code(401);

    // tell the user access denied  & s
    echo json_encode(array(
        "message" => "Access denied.",
        "error" => $e->getMessage()
    ));
}
```

7.6 Show error message if JWT is empty

If JWT is empty, it means access is also denied. We need to return a response code of 401 and tell the user access is denied.

Replace `// error if jwt is empty will be here` comment of `validate_token.php` file with the following code.

```
// show error message if jwt is empty
else{

    // set response code
    http_response_code(401);

    // tell the user access denied
    echo json_encode(array("message" =>
}
?>
```

7.7 Output

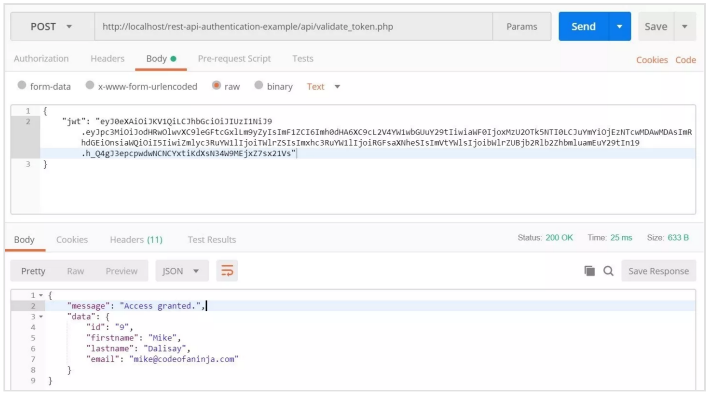
To **test for successful access**, enter the following request URL.

```
http://localhost/rest-api-authentication-example/api/validate_token.php
```

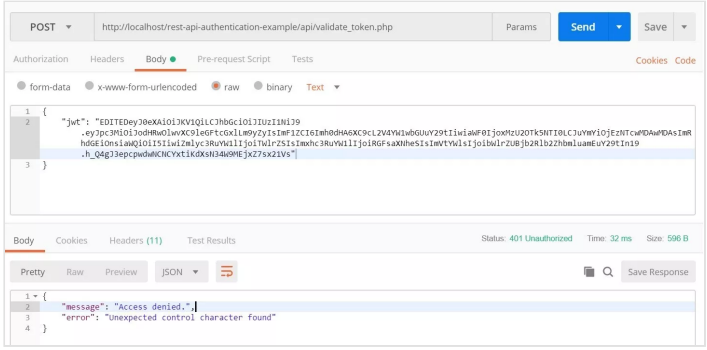
Enter the JSON Web Token we retrieved earlier. The JSON web token below is different from yours. Make sure your JWT was generated in your machine.

```
{
    "jwt": "eyJ0eXAiOiJKV1QiLCJhbGciOiJ
}
```

It should look like this on POSTMAN.



To **test for failed access**, just put a word "EDITED" on your JWT. This will make JWT wrong. It will result in a denied access. It should look like this.



8.0 CREATE API FOR USER ACCOUNT

8.1 Create a file for updating user account

This file will return an output in JSON format and will accept requests from specified URL. We'll set the correct headers.

Open `api` folder. Create `update_user.php` file. Place the following code.

```
<?php
// required headers
header("Access-Control-Allow-Origin: *"
header("Content-Type: application/json;
header("Access-Control-Allow-Methods: P
header("Access-Control-Max-Age: 3600");
header("Access-Control-Allow-Headers: C

// files for decoding jwt will be here
```

8.2 Include files to decode JWT

The code below shows the inclusion of the necessary files to decode the given JSON web token.

Replace `// files for decoding jwt will be here` comment of `update_user.php` file with the following code.

```
// required to encode json web token
include_once 'config/core.php';
include_once 'libs/php-jwt-master/src/B
include_once 'libs/php-jwt-master/src/B
include_once 'libs/php-jwt-master/src/S
include_once 'libs/php-jwt-master/src/J
use \Firebase\JWT\JWT;

// database connection will be here
```

8.3 Connect to database and user table

We will need to update user information on the database. That's why we need to get a database connection.

Replace `// database connection will be here` comment of `update_user.php` file with the following code.

```
// files needed to connect to database
include_once 'config/database.php';
include_once 'objects/user.php';

// get database connection
$database = new Database();
$db = $database->getConnection();

// instantiate user object
$user = new User($db);

// retrieve given jwt here
```

8.4 Retrieve given JWT

The code below shows how to get the value of given JSON web token.

Replace `// retrieve given jwt here` comment of `update_user.php` file with the following code.

```
// get posted data
$data = json_decode(file_get_contents('

// get jwt
$jwt=isset($data->jwt) ? $data->jwt : '

// decode jwt here
```

8.5 Decode JWT if it exists

Check if a JWT is given. If true, decode it inside a `try` block.

Replace `// decode jwt here` comment of `update_user.php` file with the following

code.

```
// if jwt is not empty
if($jwt){

    // if decode succeed, show user det
    try {

        // decode jwt
        $decoded = JWT::decode($jwt, $k

        // set user property values her
    }

    // catch failed decoding will be he
}

// error message if jwt is empty will b
```

8.6 Show error message if decoding fails

If decoding JWT fails, we need to set a response code of 401, tell the user access is denied and show information about the error.

Replace `// catch failed decoding will be here` comment of `update_user.php` file with the following code.

```
// if decode fails, it means jwt is inv
catch (Exception $e){

    // set response code
    http_response_code(401);

    // show error message
    echo json_encode(array(
        "message" => "Access denied.",
        "error" => $e->getMessage()
    ));
}
```

8.7 Set user property values

We need to set the submitted data (through the HTML form) to the `user` object properties.

Replace `// set user property values here` comment of `update_user.php` file with the following code.

```
// set user property values
$user->firstname = $data->firstname;
$user->lastname = $data->lastname;
$user->email = $data->email;
$user->password = $data->password;
$user->id = $decoded->data->id;

// update user will be here
```

8.8 Use the update() method

One the code below, we use the user object's `create()` method. If it returns `true`, it means user was updated. If it returns `false`, the system is unable to update the user information.

Replace `// update user will be here` comment of `update_user.php` file with the following code.

```
// update the user record
if($user->update()){
    // regenerate jwt will be here
}

// message if unable to update user
else{
    // set response code
    http_response_code(401);

    // show error message
    echo json_encode(array("message" =>
}
```

8.9 Add update() method in user class

The code below shows the `UPDATE` query, data sanitation and binding.

If a password was typed in the HTML form, we use the built-in `password_hash()` method to secure the user's password on the database.

If the execution is a success, the user information will be updated on the database.

Replace the `// update() method will be here` comment of `api/objects/user.php` file with the following code.

```
// update a user record
public function update(){

    // if password needs to be updated
    $password_set=!empty($this->passwor

    // if no posted password, do not up
    $query = "UPDATE " . $this->table_r
            SET
                firstname = :firstname,
                lastname = :lastname,
                email = :email
                {$password_set}
            WHERE id = :id";

    // prepare the query
    $stmt = $this->conn->prepare($query

    // sanitize
    $this->firstname=htmlspecialchars(s
    $this->lastname=htmlspecialchars(st
    $this->email=htmlspecialchars(strip

    // bind the values from the form
    $stmt->bindParam(':firstname', $thi
    $stmt->bindParam(':lastname', $this
    $stmt->bindParam(':email', $this->e

    // hash the password before saving
    if(!empty($this->password)){
        $this->password=htmlspecialchar
        $password_hash = password_hash(
        $stmt->bindParam(':password', $
    }

    // unique ID of record to be edited
    $stmt->bindParam(':id', $this->id);

    // execute the query
    if($stmt->execute()){
        return true;
    }

    return false;
}
```

8.10 Re-generate JWT

We need to re-generate or get a new JSON Web Token especially if a user information was changed. The code below does that and it sets a response code of 200 and tell the user that the information was updated.

Replace the `// regenerate jwt will be here` comment of `update_user.php` file with the following code.

```
// we need to re-generate jwt because u
$token = array(
   "iss" => $iss,
   "aud" => $aud,
   "iat" => $iat,
   "nbf" => $nbf,
   "data" => array(
       "id" => $user->id,
       "firstname" => $user->firstname,
       "lastname" => $user->lastname,
       "email" => $user->email
   )
);
$jwt = JWT::encode($token, $key);

// set response code
http_response_code(200);

// response in json format
echo json_encode(
        array(
            "message" => "User was upda
            "jwt" => $jwt
        )
    );
```

8.11 Show error message if JWT is empty

We need to tell the user that access is denied if JWT does not exist. We set a response code of 401 as well.

Replace the `// error message if jwt is empty will be here` comment of `update_user.php` file with the following code.

```
// show error message if jwt is empty
else{
```

```
    // set response code
    http_response_code(401);

    // tell the user access denied
    echo json_encode(array("message" =>
}
?>
```

8.12 Output

To **test for successful user update**, enter the following as request URL on POSTMAN.

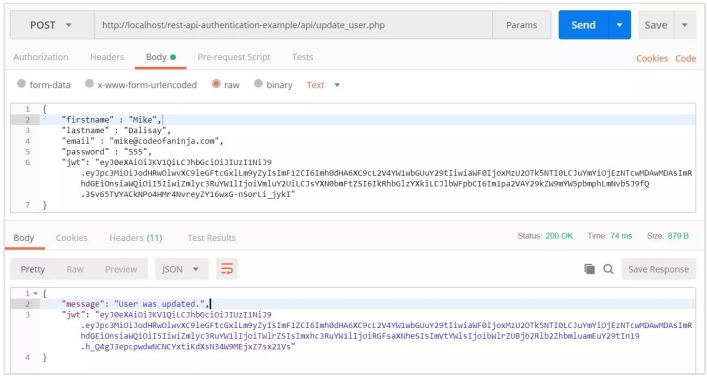
```
http://localhost/rest-api-authentication-example/api/update_user.php
```

On the body section, enter new user information with the JSON Web Token we retrieved earlier.

The JSON web token below is different from yours. Make sure your JWT was generated in your machine.

```
{
    "firstname" : "Mike",
    "lastname" : "Dalisay",
    "email" : "mike@codeofaninja.com",
    "password" : "555",
    "jwt": "eyJ0eXAiOiJKV1QiLCJhbGciOiJ
}
```

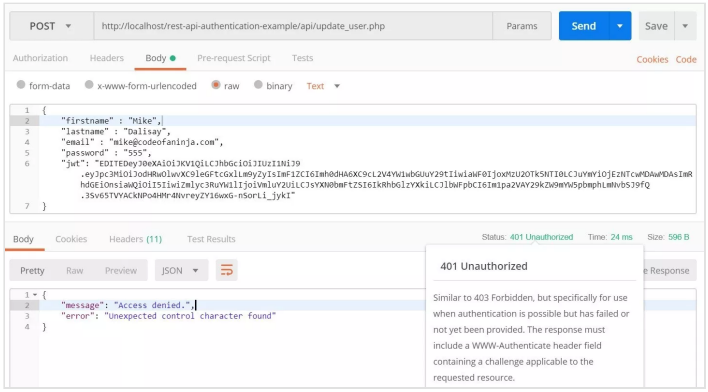
It should look like this on POSTMAN.



As you can see on the image above, it generates a new JWT and it will be stored in

the client application. We can use the new information on the app interface later.

To **test for failed user update**, you can just add the word `EDITED` on the submitted JWT or just remove the JWT. It should look like the following.



[https://i0.wp.com/www.codeofaninja.com/wp-content/uploads/2018/09/unable-to-update-user-access-denied.jpg?ssl=1]

9.0 CREATE INTERFACE FOR USER REGISTRATION

9.1 Create index page

We will use the APIs we created earlier on a simple Single-Page Application (SPA) created using HTML, CSS and JavaScript.

All of the essential codes will be in this single `index.html` file.

Open `rest-api-authentication-example` folder. Create `index.html` file. Place the following code.

```
<!doctype html>
<html lang="en">
    <head>
        <!-- Required meta tags -->
```

```
        <meta charset="utf-8" />
        <meta name="viewport" content="

        <title>Rest API Authentication

        <!-- CSS links will be here -->

    </head>
<body>

<!-- navigation bar will be here -->

<!-- script links will be here -->

</body>
</html>
```

9.2 Add navigation bar

The navigation bar is where the menus like home page, account page, login page, logout and sign up page can be clicked or triggered.

Replace the `<!-- navigation bar will be here -->` comment of `index.html` file with the following code.

```
<!-- navbar -->
<nav class="navbar navbar-expand-md nav
    <a class="navbar-brand" href="#">Na
    <button class="navbar-toggler" type
        <span class="navbar-toggler-icc
    </button>
    <div class="collapse navbar-collaps
        <div class="navbar-nav">
            <a class="nav-item nav-link
            <a class="nav-item nav-link
            <a class="nav-item nav-link
            <a class="nav-item nav-link
            <a class="nav-item nav-link
        </div>
    </div>
</nav>
<!-- /navbar -->

<!-- content section will be here -->
```

9.3 Add content section

The content section is where the contents like HTML forms and message prompts will be rendered.

Replace the `<!-- content section will be here -->` comment of `index.html` file with the following code.

```
<!-- container -->
<main role="main" class="container star

    <div class="row">
        <div class="col">

            <!-- where prompt / message
            <div id="response"></div>

            <!-- where main content wil
            <div id="content"></div>
        </div>
    </div>

</main>
<!-- /container -->
```

9.4 Add Bootstrap 4 and custom CSS links

We are using Bootstrap 4 [https://getbootstrap.com/] to make the user interface look good. We will use the CDN link so that we won't have to download the whole library.

We will see the use of custom CSS file on the next section.

Replace the `<!-- CSS links will be here -->` comment of `index.html` file with the following code.

```
<!-- Bootstrap 4 CSS and custom CSS -->
<link rel="stylesheet" href="https://ma
<link rel="stylesheet" type="text/css"
```

9.5 Create custom CSS file

We use the custom CSS for any look & feel customization we want to implement.

Open `rest-api-authentication-example` folder. Create `custom.css` file.

Place the following code.

```
body { padding-top: 5rem; }
.starter-template { padding: 3rem 1.5re
#logout{ display:none; }
```

9.6 Add jQuery and Bootstrap 4 script links

In this tutorial, we use the jQuery library to render the interface and make HTTP requests.

To make Bootstrap 4 work, we need to include its own JavaScript as well.

Replace the `<!-- script links will be here -->` comment of `index.html` file with the following code.

```
<!-- jQuery & Bootstrap 4 JavaScript li
<script src="https://code.jquery.com/jq
<script src="https://cdnjs.cloudflare.c
<script src="https://maxcdn.bootstrapcd

<!-- jquery scripts will be here -->
```

9.7 Show a sign up HTML form

When you click the Sign Up menu on the navigation bar, it will show a sign up or registration form.

The code below shows the `click` trigger and the HTML form.

Replace the `<!-- jquery scripts will be here -->` comment of `index.html` file with the following code.

```
<script>
// jQuery codes
$(document).ready(function(){
    // show sign up / registration form
    $(document).on('click', '#sign_up',

        var html = `
            <h2>Sign Up</h2>
            <form id='sign_up_form'>
```

```
                <div class="form-group"
                    <label for="firstna
                    <input type="text"
                </div>

                <div class="form-group"
                    <label for="lastnam
                    <input type="text"
                </div>

                <div class="form-group"
                    <label for="email">
                    <input type="email"
                </div>

                <div class="form-group"
                    <label for="passwor
                    <input type="passwo
                </div>

                <button type='submit' c
            </form>
            `;

        clearResponse();
        $('#content').html(html);
    });

    // trigger when registration form i

    // show login form trigger will be

    // clearResponse() will be here
});
</script>
```

9.8 Trigger when sign up form is submitted

We need to process the form data when it is submitted.

Replace the `trigger when registration form is submitted here` comment of `index.html` file with the following code.

```
// trigger when registration form is su
$(document).on('submit', '#sign_up_form

    // get form data
    var sign_up_form=$(this);
    var form_data=JSON.stringify(sign_u

    // submit form data to api
    $.ajax({
        url: "api/create_user.php",
```

```
        type : "POST",
        contentType : 'application/jsor
        data : form_data,
        success : function(result) {
            // if response is a success
            $('#response').html("<div c
            sign_up_form.find('input').
        },
        error: function(xhr, resp, text
            // on error, tell the user
            $('#response').html("<div c
        }
    });

    return false;
});
```

9.9 Remove any prompt messages

The `clearResponse()` method was used on the previous section. Its only purpose is to remove any prompt messages that may have been displayed on the screen.

Replace the `// clearResponse() will be here` comment of `index.html` file with the following code.

```
// remove any prompt messages
function clearResponse(){
    $('#response').html('');
}

// showLoginPage() will be here

// serializeObject will be here
```

9.10 Add serializeObject function

The serializeObject function will convert form data to JSON format. We need this function to send values from an HTML form to the API.

Replace the `// serializeObject will be here` comment of `index.html` file with the following code.

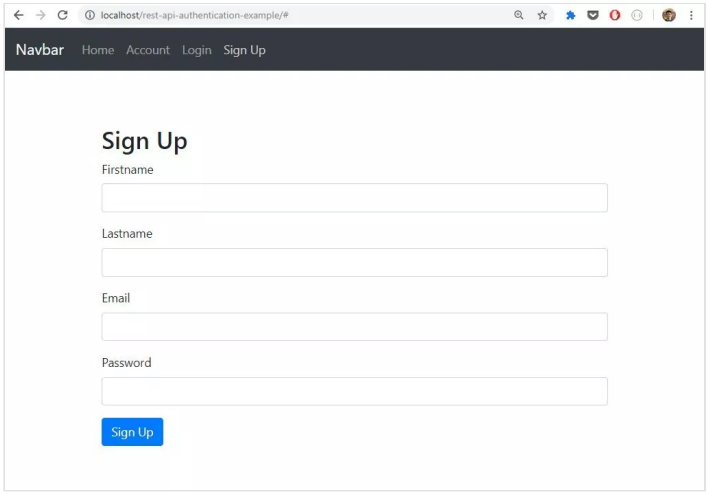
```
// function to make form values to jsor
$.fn.serializeObject = function(){

    var o = {};
```

```
    var a = this.serializeArray();
    $.each(a, function() {
        if (o[this.name] !== undefined)
            if (!o[this.name].push) {
                o[this.name] = [o[this.
            }
            o[this.name].push(this.valu
        } else {
            o[this.name] = this.value |
        }
    });
    return o;
};
```

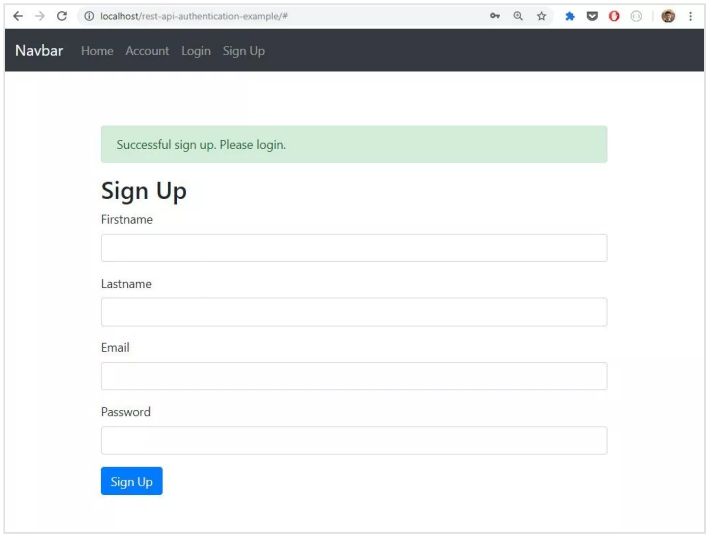
9.11 Output

When user click the `Sign Up` link on the navigation bar.



[https://i2.wp.com/www.codeofaninja.com/wp-content/uploads/2018/09/3-sign-up.jpg?ssl=1]

After the user filled out and submitted the form.



[https://i1.wp.com/www.codeofaninja.com/wp-content/uploads/2018/09/4-successful-sign-up.jpg?ssl=1]

10.0 CREATE INTERFACE FOR USER LOGIN

10.1 Trigger when login menu was clicked

When you click the `Login` menu on the navigation bar, it will show a login form.

The code below shows the `click` trigger and `showLoginPage();` function to show a login form.

Replace the `// show login form trigger will be here` comment of `index.html` file with the following code.

```
// show login form
$(document).on('click', '#login', funct
    showLoginPage();
});

// login form submit trigger will be he
```

10.2 Show login HTML form

The function below shows the HTML form for users to login.

Replace the `// showLoginPage() will be here` comment of `index.html` file with the following code.

```
// show login page
function showLoginPage(){

    // remove jwt
    setCookie("jwt", "", 1);

    // login page html
    var html = `
        <h2>Login</h2>
        <form id='login_form'>
            <div class='form-group'>
                <label for='email'>Emai
                <input type='email' cla
            </div>

            <div class='form-group'>
                <label for='password'>P
                <input type='password'
            </div>

            <button type='submit' class
        </form>
        `;

    $('#content').html(html);
    clearResponse();
    showLoggedOutMenu();
}

// setCookie() will be here

// showLoggedOutMenu() will be here
```

10.3 Add setCookie() function

The `setCookie()` function will help us store JWT on the cookie.

Replace the `// setCookie() will be here` comment of `index.html` file with the following code.

```
// function to set cookie
function setCookie(cname, cvalue, exday
    var d = new Date();
    d.setTime(d.getTime() + (exdays*24*
```

```
    var expires = "expires="+ d.toUTCSt
    document.cookie = cname + "=" + cva
}
```

10.4 Change menu appearance

The `showLoggedOutMenu()` function was used in the previous section.

This function will make the menu look like the options for a logged-out user.

Replace the `// showLoggedOutMenu() will be here` comment of `index.html` file with the following code.

```
// if the user is logged out
function showLoggedOutMenu(){
    // show login and sign up from navb
    $("#login, #sign_up").show();
    $("#logout").hide();
}

// showHomePage() function will be here
```

10.5 Trigger when login form is submitted

The code below shows a `submit` trigger for the login form.

It gets the data from the form and store it in the `form_data` variable.

Replace the `// login form submit trigger will be here` comment of `index.html` file with the following code.

```
// trigger when login form is submitted
$(document).on('submit', '#login_form',

    // get form data
    var login_form=$(this);
    var form_data=JSON.stringify(login_

    // http request will be here

    return false;
});
```

```
// trigger to show home page will be he
```

10.6 Create an HTTP request

The code below shows how we make an HTTP request, specifically an AJAX request to verify if the submitted email and password is valid.

If it is valid, we will save the JWT to `localStorage`, show the home page and tell the user it was a successful login.

Replace the `// http request will be here` comment of `index.html` file with the following code.

```
// submit form data to api
$.ajax({
    url: "api/login.php",
    type : "POST",
    contentType : 'application/json',
    data : form_data,
    success : function(result){

        // store jwt to cookie
        setCookie("jwt", result.jwt, 1)

        // show home page & tell the us
        showHomePage();
        $('#response').html("<div class

    },
    // error response will be here
});
```

10.7 Show error if HTTP request fails

If the submitted email and password is invalide, we tell the user login failed and empty the login form.

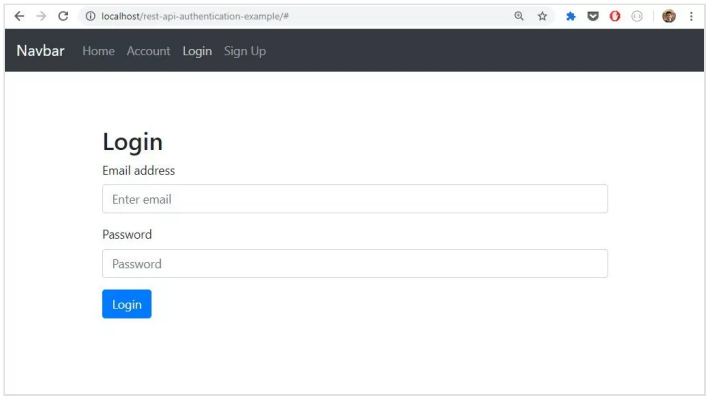
Replace the `// error response will be here` comment of `index.html` file with the following code.

```
error: function(xhr, resp, text){
    // on error, tell the user login ha
    $('#response').html("<div class='al
```

```
    login_form.find('input').val('');
}
```

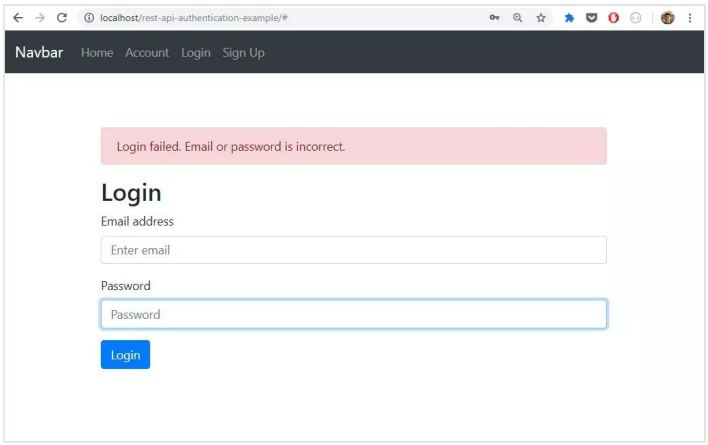
10.8 Output

If the user clicks the `Login` menu on the navigation bar.



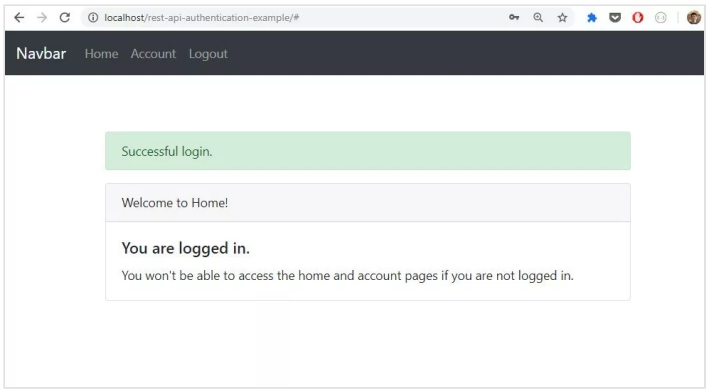
[https://i1.wp.com/www.codeofaninja.com/wp-content/uploads/2018/09/5-login.jpg?ssl=1]

If the user entered an invalid email or password.



[https://i1.wp.com/www.codeofaninja.com/wp-content/uploads/2018/09/7-failed-login-screen.jpg?ssl=1]

If the user entered a valid email and password.



[https://i1.wp.com/www.codeofaninja.com/wp-content/uploads/2018/09/6-successful-login-screen.jpg?ssl=1]

11.0 CREATE INTERFACE FOR HOME PAGE

11.1 Add trigger to show home page

The code below shows a `click` trigger with `showHomePage();` function.

Replace the `// trigger to show home page will be here` comment of `index.html` file with the following code.

```
// show home page
$(document).on('click', '#home', functi
    showHomePage();
    clearResponse();
});

// trigger to show account form will be
```

11.2 Verify if JWT is valid

On the `showHomePage()` function, we need to validate the stored JWT before showing the home page HTML.

Replace the `// showHomePage() function will be here` comment of `index.html`

file with the following code.

```
// show home page
function showHomePage(){

    // validate jwt to verify access
    var jwt = getCookie('jwt');
    $.post("api/validate_token.php", JS

        // home page html will be here
    })

    // show login page on error will be
}

// getCookie() will be here

// showLoggedInMenu() will be here
```

11.3 Add getCookie() function

The `getCookie()` function will help us read the JWT we stored earlier.

Replace the `// getCookie() will be here` comment of `index.html` file with the following code.

```
// get or read cookie
function getCookie(cname){
    var name = cname + "=";
    var decodedCookie = decodeURICompor
    var ca = decodedCookie.split(';');
    for(var i = 0; i <ca.length; i++) {
        var c = ca[i];
        while (c.charAt(0) == ' '){
            c = c.substring(1);
        }

        if (c.indexOf(name) == 0) {
            return c.substring(name.ler
        }
    }
    return "";
}
```

11.4 Add home page HTML

If JWT is valid, we show the home page HTML and call the `showLoggedInMenu()` function.

Replace the `// home page html will be here` comment of `index.html` file with the following code.

```
// if valid, show homepage
var html = `
    <div class="card">
        <div class="card-header">Welcom
        <div class="card-body">
            <h5 class="card-title">You
            <p class="card-text">You wc
        </div>
    </div>
    `;

$('#content').html(html);
showLoggedInMenu();
```

11.5 Set logged-in menu

The `showLoggedInMenu()` function will change the menu options to look like a menu for a logged-in user.

Replace the `showLoggedInMenu() will be here` comment of `index.html` file with the following code.

```
// if the user is logged in
function showLoggedInMenu(){
    // hide login and sign up from navb
    $("#login, #sign_up").hide();
    $("#logout").show();
}

// showUpdateAccountForm() will be here
```

11.6 Show login page if JWT is invalid

If JWT is invalid, we will show the login page and ask the user to login.

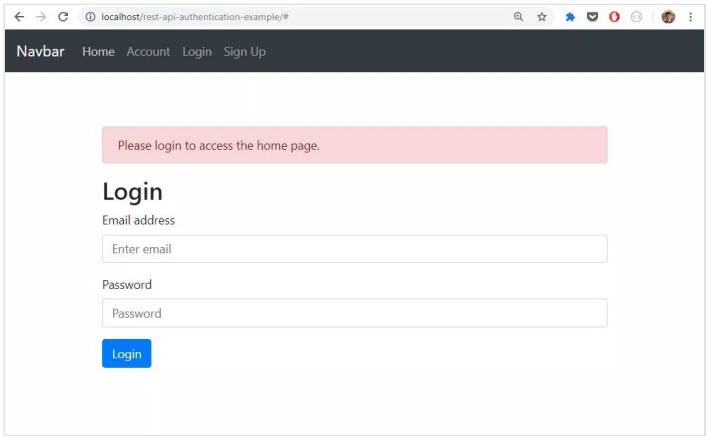
Replace the `// show login page on error will be here` comment of `index.html` file with the following code.

```
// show login page on error
.fail(function(result){
    showLoginPage();
```

```
    $('#response').html("<div class='al
});
```

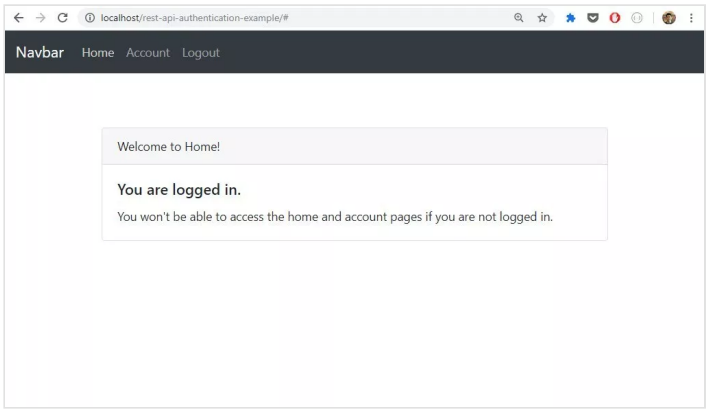
11.7 Output

If a logged-out user clicked the `Home` menu on the navigation bar.



[https://i0.wp.com/www.codeofaninja.com/wp-content/uploads/2018/09/1-home-page-not-logged-in.jpg?ssl=1]

If a logged-in user clicked the `Home` menu on the navigation bar.



[https://i0.wp.com/www.codeofaninja.com/wp-content/uploads/2018/09/7-home-page-logged-in.jpg?ssl=1]

12.0 CREATE INTERFACE FOR ACCOUNT PAGE

12.1 Add trigger to show account form

The code below shows a `click` trigger with `showUpdateAccountForm();` function.

Replace the `// trigger to show account form will be here` comment of `index.html` file with the following code.

```
// show update account form
$(document).on('click', '#update_accour
    showUpdateAccountForm();
});

// trigger for updating user account wi
```

12.2 Verify if JWT is valid

We need the `showUpdateAccountForm()` function to render to HTML form for updating a user account.

First, we need to verify if JWT is valid. We use the `getCookie('jwt');` function to get the JWT and send it to `validate_token.php` via jQuery `$.post` method.

Replace the `// showUpdateAccountForm() will be here` comment of `index.html` file with the following code.

```
function showUpdateAccountForm(){
    // validate jwt to verify access
    var jwt = getCookie('jwt');
    $.post("api/validate_token.php", JS

        // html form for updating user
    })

    // error message when jwt is invali
}
```

12.3 Show account form if JWT is valid

If JWT is valid, we will show the HTML form using the code below.

Replace the `// html form for updating user account will be here` comment of `index.html` file with the following code.

```
// if response is valid, put user detai
var html = `
        <h2>Update Account</h2>
        <form id='update_account_form'>
            <div class="form-group">
                <label for="firstname">
                <input type="text" clas
            </div>

            <div class="form-group">
                <label for="lastname">L
                <input type="text" clas
            </div>

            <div class="form-group">
                <label for="email">Emai
                <input type="email" cla
            </div>

            <div class="form-group">
                <label for="password">P
                <input type="password"
            </div>

            <button type='submit' class
                Save Changes
            </button>
        </form>
    `;

clearResponse();
$('#content').html(html);
```

12.4 Show login page if JWT is invalid

If JWT is invalid, we will logout the user and ask him to login.

Replace the `// error message when jwt is invalid will be here` comment of `index.html` file with the following code.

```
// on error/fail, tell the user he neec
.fail(function(result){
    showLoginPage();
    $('#response').html("<div class='al
});
```

12.5 Add a trigger for updating user account

If the submit button was clicked, we will use the code below to catch that trigger.

We will get the form handle and JWT as well.

Replace the `// trigger for updating user account will be here` comment of `index.html` file with the following code.

```
// trigger when 'update account' form i
$(document).on('submit', '#update_accou

    // handle for update_account_form
    var update_account_form=$(this);

    // validate jwt to verify access
    var jwt = getCookie('jwt');

    // get form data and jwt here

    return false;
});

// trigger to logout will be here
```

12.6 Get form data and JWT

On the code below, we get the form values and add the JWT to it. We convert the form values to JSON via `stringify()` function so that it can be sent to the API.

Replace the `// get form data and jwt here` comment of `index.html` file with the following code.

```
// get form data
var update_account_form_obj = update_ac

// add jwt on the object
update_account_form_obj.jwt = jwt;

// convert object to json string
var form_data=JSON.stringify(update_acc

// send data to api here
```

12.7 Send data to API

We send the form values to update_user.php using jQuery AJAX method. If the response is successful, we tell the user his account was updated.

We store the new JWT to `localStorage` as well.

Replace the `// send data to api here` comment of `index.html` file with the following code.

```
// submit form data to api
$.ajax({
    url: "api/update_user.php",
    type : "POST",
    contentType : 'application/json',
    data : form_data,
    success : function(result) {

        // tell the user account was up
        $('#response').html("<div class

        // store new jwt to coookie
        setCookie("jwt", result.jwt, 1)
    },

    // errors will be handled here
});
```

12.8 Show error message

If the system is unable to update the user, we tell the user about that.

If JWT is invalid and access is denied, we logout the user and ask him to login.

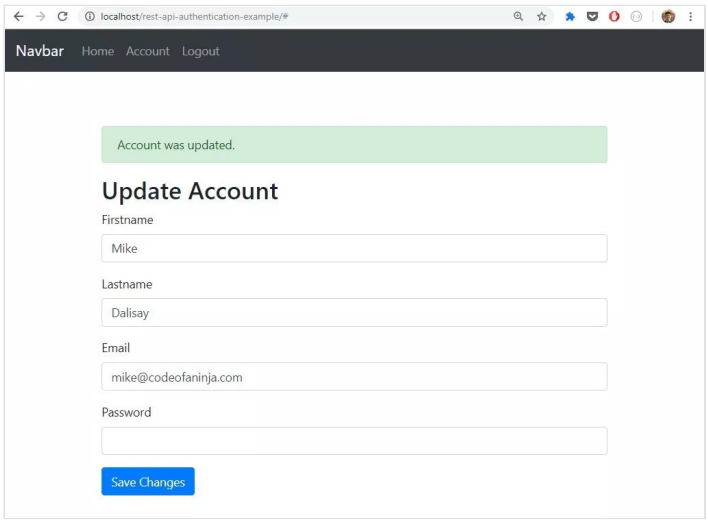
Replace the `// errors will be handled here` comment of `index.html` file with the following code.

```
// show error message to user
error: function(xhr, resp, text){
    if(xhr.responseJSON.message=="Unabl
        $('#response').html("<div class
    }
```

```
    else if(xhr.responseJSON.message=='
        showLoginPage();
        $('#response').html("<div class
    }
}
```

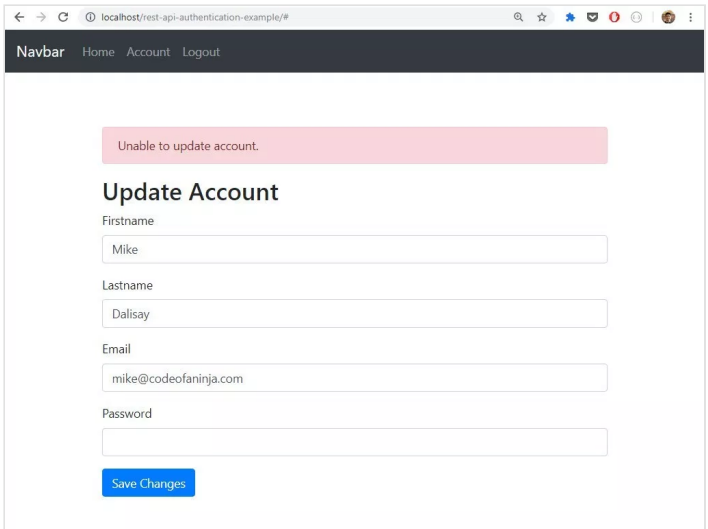
12.9 Output

If user account was updated successfully.



[https://i1.wp.com/www.codeofaninja.com/wp-content/uploads/2018/09/9-update-account-success.jpg?ssl=1]

If there was a problem when updating user account.



[https://i0.wp.com/www.codeofaninja.com/wp-content/uploads/2018/09/9-update-account-failed.jpg?ssl=1]

13.0 ADD JAVASCRIPT FOR USER LOGOUT

13.1 Add a trigger to logout

The `click` trigger below is used when the user click the `Logout` link on the menu.

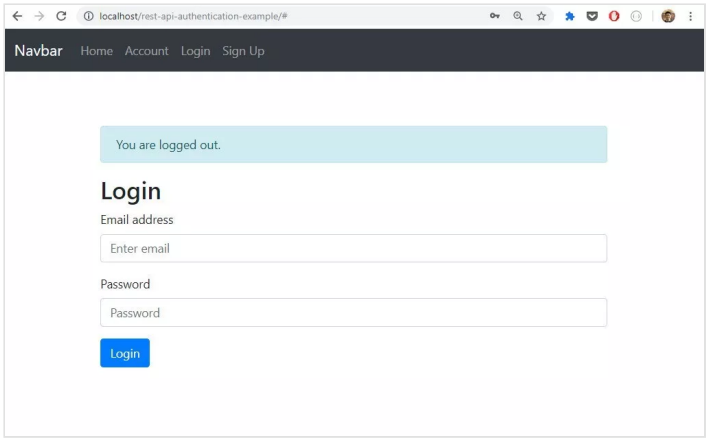
We use the `showLoginPage();` method to logout the user. We tell he is logged out as well.

Replace the `// trigger to logout will be here` comment of `index.html` file with the following code.

```
// logout the user
$(document).on('click', '#logout', func
    showLoginPage();
    $('#response').html("<div class='al
});
```

13.2 Output

If the user clicked the `Logout` link on the menu.



[https://i0.wp.com/www.codeofaninja.com/wp-content/uploads/2018/09/10-logout.jpg?ssl=1]

14.0 DOWNLOAD SOURCE CODES

We highly recommend for you to follow and study our well-detailed, step-by-step tutorial above first. Nothing beats experience when it comes to learning.

But we believe you will learn faster if you'll see the final source code as well. We consider it as your additional guide.

Imagine the value or skill upgrade it can bring you. The additional income you can get from your work, projects or business. The precious time you save. Isn't that what you want?

14.1 Download LEVEL 1 source code

FEATURES	LEVEL 1
API for user registration / sign up	YES
API for user login	YES
API for JWT validation	YES
API for updating user account	YES
Sign up page / registration form	YES
HTML5 validation for registration form	YES
Tell the user if sign up is successful	YES
Login using email and password	YES
Tell the user if login failed	YES
Tell the user if successfully logged in	YES

Restricted access to home page	YES
Restricted access to account page	YES
Show home page when logged in	YES
Show accounts page when logged in	YES
Update user information	YES
Tell the user if updating the account failed	YES
Logout user	YES
FREE email support for 3 months	YES
Source code updates via email	YES
○ LEVEL 1 Source Code	$30.00
◉ LEVEL 2 Source Code	$60.00
DOWNLOAD NOW [#]	

14.2 Download LEVEL 2 source code

FEATURES	LEVEL 2
All features of LEVEL 1 source code	YES
User access level	YES
Admin access level	YES
Admin can create user	YES
Admin can read different user information	YES
Admin can update user information	YES
Admin can delete user	YES

Admin can read users with pagination	YES
Admin can search users	YES
API for creating a user	YES
API for reading users list (with pagination)	YES
API for reading user information	YES
API for updating a user	YES
API for deleting a user	YES
Tell the user if a request fails or succeeds.	YES
Validate JWT for every HTTP request	YES
FREE email support for 6 months	YES
Source code updates via email	YES
○ LEVEL 1 Source Code	$30.00
◉ LEVEL 2 Source Code	$60.00

DOWNLOAD NOW [#]

15.0 WHAT'S NEXT?

We will learn how to create, read, update and delete database records (with user interface) on our AJAX CRUD Tutorial [https://www.codeofaninja.com/2015/06/php-crud-with-ajax-and-oop.html] .

If you want to receive new and updated high-quality tutorials, please subscribe [https://www.codeofaninja.com/subscribe] for free.

16.0 RELATED TUTORIALS

GETTING STARTED

Learn the basics of web programming.

How to Run a PHP Script? [https://www.codeofaninja.com/2013/06/how to-run-a-php-script.html]

Bootstrap Tutorial for Beginners [https://www.codeofaninja.com/2014/05/boot tutorial-beginners-step-step.html]

jQuery Tutorial for Beginners [https://www.codeofaninja.com/2013/10/jque step-by-step-tutorial-for-beginners.html]

jQuery UI Tutorial for Beginners [https://www.codeofaninja.com/2013/10/jque ui-tutorial-for-beginners.html]

PHP PROGRAMMING TUTORIALS

Start something awesome with PHP!

PHP CRUD Tutorial for Beginners [https://www.codeofaninja.com/2011/12/php and-mysql-crud-tutorial.html]

PHP OOP CRUD Tutorial [https://www.codeofaninja.com/2014/06/php object-oriented-crud-example-oop.html]

PHP Login Script with Session [https://www.codeofaninja.com/2013/03/php login-script.html]

PHP Shopping Cart Tutorial [https://www.codeofaninja.com/2015/08/simp php-mysql-shopping-cart-tutorial.html]

PHP Shopping Cart Tutorial 2.0 [https://www.codeofaninja.com/2013/04/shop cart-in-php.html]

REST API TUTORIALS

Welcome to the new world of web development!

PHP REST API Tutorial [https://www.codeofaninja.com/2017/02/crea simple-rest-api-in-php.html]

PHP REST API Authentication Example [https://www.codeofaninja.com/2018/09/rest-api-authentication-example-php-jwt-tutorial.html]

AJAX CRUD Tutorial [https://www.codeofaninja.com/2015/06/php· crud-with-ajax-and-oop.html]

PHP WEB APP SOURCE CODES

Scripts that will help you build a web app.

PHP SHOPPING CART SYSTEM [https://www.codeofaninja.com/2016/04/php· shopping-cart-source-code-download.html]

Download By Modules:

PHP Login & Registration Module [https://www.codeofaninja.com/2016/05/php· login-system-tutorial.html]

PHP Shopping Cart Module [https://www.codeofaninja.com/2016/07/php· online-shopping-cart-source-code.html]

PHP Product Catalog Module [https://www.codeofaninja.com/2016/07/php· product-catalog-script.html]

PHP Content Management Module [https://www.codeofaninja.com/2016/07/php· web-page-content-management-module.html]

PHP Contact Form Module [https://www.codeofaninja.com/2016/07/php· contact-form-messages-module.html]

PHP PayPal Integration Module [https://www.codeofaninja.com/2016/04/payp integration-in-php.html]

MORE SCRIPTS

More code examples that can be useful for you!

Shopping Cart Tutorial using COOKIES [https://www.codeofaninja.com/2014/09/php-shopping-cart-tutorial-using-cookies.html]

Extra Tutorials [https://www.codeofaninja.com/extras]

17.0 NOTES

#1 Found An Issue?

If you found a problem with this code, please write a comment below. Please be descriptive about your issue. Please provide the error messages, screenshots (or screencast) and your test URL. Thanks!

Before you write a comment, remember to read this guide [https://www.codeofaninja.com/how-to-write-a-great-comment-on-codeofaninja-com] and our code of conduct [https://www.codeofaninja.com/code-of-conduct] .

#2 Become a true Ninja!

We constantly add new tutorials and improve our existing tutorials and source codes. Be one of the first to know an update by subscribing to our FREE newsletter. CLICK HERE TO SUBSCRIBE FOR FREE! [https://www.codeofaninja.com/subscribe]

#3 Thank You!

Please note that this post is not yet in its final form. We are going to update this post so it will be perfect in the future.

If you have a friend or know someone who needs this REST API Authentication Example in PHP & JWT, please share this page to them! I know you will help them a lot by doing it. Thanks!

Before you write a comment, please read this guide and our code of conduct.