REST API Authentication Example in PHP – JWT Tutorial

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.

**Contents**[[hide](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/)]

* [1 Project Overview](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Project_Overview)
  + [1.1 What is JWT?](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#What_is_JWT)
  + [1.2 JWT simple analogy](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#JWT_simple_analogy)
  + [1.3 What does a JWT look like?](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#What_does_a_JWT_look_like)
  + [1.4 JWT vs OAuth](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#JWT_vs_OAuth)
* [2 Final output](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Final_output)
  + [2.1 LEVEL 1 source code output](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#LEVEL_1_source_code_output)
  + [2.2 LEVEL 2 source code output](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#LEVEL_2_source_code_output)
* [3 File Structure](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#File_Structure)
* [4 Setup the Database](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Setup_the_Database)
  + [4.1 Create a database](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Create_a_database)
  + [4.2 Create a table](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Create_a_table)
  + [4.3 Create a directory for configuration](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Create_a_directory_for_configuration)
  + [4.4 Create a database connection file](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Create_a_database_connection_file)
* [5 Create API for user registration](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Create_API_for_user_registration)
  + [5.1 Create a file for creating a user](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Create_a_file_for_creating_a_user)
  + [5.2 Connect to database and user table](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Connect_to_database_and_user_table)
  + [5.3 Assign submitted data to object properties](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Assign_submitted_data_to_object_properties)
  + [5.4 Use the create() method](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Use_the_create_method)
  + [5.5 Create the user object class](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Create_the_user_object_class)
  + [5.6 Add a create() method](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Add_a_create_method)
  + [5.7 Output](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Output)
* [6 Create API for user login](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Create_API_for_user_login)
  + [6.1 Create a file for user login](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Create_a_file_for_user_login)
  + [6.2 Connect to database and user table](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Connect_to_database_and_user_table-2)
  + [6.3 Check if email exists](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Check_if_email_exists)
  + [6.4 Add emailExists() method](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Add_emailExists_method)
  + [6.5 Include files to encode JWT](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Include_files_to_encode_JWT)
  + [6.6 Generate JSON web token](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Generate_JSON_web_token)
  + [6.7 Tell the user login failed](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Tell_the_user_login_failed)
  + [6.8 Create core configuration file](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Create_core_configuration_file)
  + [6.9 Download PHP-JWT from GitHub](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Download_PHP-JWT_from_GitHub)
  + [6.10 Output](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Output-2)
* [7 Create API for JWT validation](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Create_API_for_JWT_validation)
  + [7.1 Create a file to validate JWT](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Create_a_file_to_validate_JWT)
  + [7.2 Include files to decode JWT](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Include_files_to_decode_JWT)
  + [7.3 Retrieve given JWT](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Retrieve_given_JWT)
  + [7.4 Decode JWT if it exists](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Decode_JWT_if_it_exists)
  + [7.5 Show error if decoding failed](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Show_error_if_decoding_failed)
  + [7.6 Show error message if JWT is empty](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Show_error_message_if_JWT_is_empty)
  + [7.7 Output](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Output-3)
* [8 Create API for user account](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Create_API_for_user_account)
  + [8.1 Create a file for updating user account](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Create_a_file_for_updating_user_account)
  + [8.2 Include files to decode JWT](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Include_files_to_decode_JWT-2)
  + [8.3 Connect to database and user table](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Connect_to_database_and_user_table-3)
  + [8.4 Retrieve given JWT](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Retrieve_given_JWT-2)
  + [8.5 Decode JWT if it exists](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Decode_JWT_if_it_exists-2)
  + [8.6 Show error message if decoding fails](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Show_error_message_if_decoding_fails)
  + [8.7 Set user property values](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Set_user_property_values)
  + [8.8 Use the update() method](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Use_the_update_method)
  + [8.9 Add update() method in user class](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Add_update_method_in_user_class)
  + [8.10 Re-generate JWT](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Re-generate_JWT)
  + [8.11 Show error message if JWT is empty](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Show_error_message_if_JWT_is_empty-2)
  + [8.12 Output](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Output-4)
* [9 Create interface for user registration](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Create_interface_for_user_registration)
  + [9.1 9.1 Create index page](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#91_Create_index_page)
  + [9.2 Add navigation bar](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Add_navigation_bar)
  + [9.3 Add content section](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Add_content_section)
  + [9.4 Add Bootstrap 4 and custom CSS links](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Add_Bootstrap_4_and_custom_CSS_links)
  + [9.5 Create custom CSS file](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Create_custom_CSS_file)
  + [9.6 Add jQuery and Bootstrap 4 script links](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Add_jQuery_and_Bootstrap_4_script_links)
  + [9.7 Show a sign up HTML form](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Show_a_sign_up_HTML_form)
  + [9.8 Trigger when sign up form is submitted](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Trigger_when_sign_up_form_is_submitted)
  + [9.9 Remove any prompt messages](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Remove_any_prompt_messages)
  + [9.10 Add serializeObject function](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Add_serializeObject_function)
  + [9.11 Output](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Output-5)
* [10 Create a login page](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Create_a_login_page)
  + [10.1 10.1 Trigger when login menu was clicked](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#101_Trigger_when_login_menu_was_clicked)
  + [10.2 Show login HTML form](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Show_login_HTML_form)
  + [10.3 Add setCookie() function](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Add_setCookie_function)
  + [10.4 Change menu appearance](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Change_menu_appearance)
  + [10.5 Output](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Output-6)
* [11 Show login responses](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Show_login_responses)
  + [11.1 Submitted form trigger](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Submitted_form_trigger)
  + [11.2 Create an HTTP request](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Create_an_HTTP_request)
  + [11.3 Show home page HTML](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Show_home_page_HTML)
  + [11.4 Add home page HTML](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Add_home_page_HTML)
  + [11.5 Add getCookie() function](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Add_getCookie_function)
  + [11.6 Show error for invalid login](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Show_error_for_invalid_login)
  + [11.7 Output](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Output-7)
* [12 On click of Home menu](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#On_click_of_Home_menu)
  + [12.1 Add trigger to show home page](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Add_trigger_to_show_home_page)
  + [12.2 Set logged-in menu](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Set_logged-in_menu)
  + [12.3 Show login page if JWT is invalid](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Show_login_page_if_JWT_is_invalid)
  + [12.4 Output](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Output-8)
* [13 Show user account page](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Show_user_account_page)
  + [13.1 Add trigger to show account form](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Add_trigger_to_show_account_form)
  + [13.2 Verify if JWT is valid](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Verify_if_JWT_is_valid)
  + [13.3 Show account form if JWT is valid](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Show_account_form_if_JWT_is_valid)
  + [13.4 Show login page if JWT is invalid](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Show_login_page_if_JWT_is_invalid-2)
  + [13.5 Add a trigger for updating a user account](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Add_a_trigger_for_updating_a_user_account)
  + [13.6 Get form data and JWT](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Get_form_data_and_JWT)
  + [13.7 Send data to API](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Send_data_to_API)
  + [13.8 Show error message](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Show_error_message)
  + [13.9 Output](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Output-9)
* [14 Add JavaScript for user logout](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Add_JavaScript_for_user_logout)
  + [14.1 Add a trigger to logout](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Add_a_trigger_to_logout)
  + [14.2 Output](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Output-10)
* [15 Download source codes](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Download_source_codes)
  + [15.1 Download LEVEL 1 source code](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Download_LEVEL_1_source_code)
  + [15.2 15.2 Download LEVEL 2 source code](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#152_Download_LEVEL_2_source_code)
* [16 What’s Next?](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#What8217s_Next)
* [17 Related Tutorials](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Related_Tutorials)
* [18 Notes](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Notes)
  + [18.1 Found An Issue?](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Found_An_Issue)
  + [18.2 Subscribe to CodeOfaNinja](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Subscribe_to_CodeOfaNinja)
  + [18.3 Thank You!](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Thank_You)
  + [18.4 Share our tutorial](https://codeofaninja.com/rest-api-authentication-example-php-jwt-tutorial/#Share_our_tutorial)

**Project Overview**

**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.

* JWTs are commonly used for organizations that have multiple applications or domains. Their users need to be authenticated (log in) once and be authorized to access multiple applications, domains, or resources of that organization.
* JWTs are also used for an application that uses multiple servers.
* In a single-server application, users can be authorized by session ID. A server will need to find (by lookup) a valid session ID on every HTTP request. Session IDs are stored on the server.
* If your application needs to scale and needs multiple servers, you cannot use session IDs for user authorization. Every HTTP request will be routed to a different server where the session ID does not exist. Users will be logged out unexpectedly.
* JWT helps solve the multi-server authorization problem by storing the authorized user information on the client-side, not on the server. JWT is sent on every HTTP request and it can be validated (by code) of any server used by the application.

You may read more [here](https://jwt.io/introduction/) and [here](https://en.wikipedia.org/wiki/JSON_Web_Token).

**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

1. **John (client)** is a student. **Michael (server)** is the professor. John sends his **test paper (login credentials)** to Michael.
2. Michael checked, validated, and **signed the test paper (JWT)**. This proves that John is a student of Michael.
3. Michael sends back the **signed test paper (JWT**) to John.
4. If John has **further questions about the test (HTTP requests)**, he can present his signed test paper to Michael.
5. Michael will **answer (response)** because he signed the test paper.
6. Now, this is the useful part. Michael is not available. **Vincent (another server)** is the professor that is available.
7. John has **further questions about the test (HTTP requests)**. Vincent needs proof that John is a student of Michael.
8. John presented his **signed test paper (JWT)**to Vincent. Vincent will **answer (response)** because it was signed by Michael.
9. If the test paper was not signed, Michael and Vincent will not answer John’s further questions.

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

The videos below will help explain the analogy.

**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.eyJpc3MiOiJodHRwOlwvXC9leGFtcGxlLm9yZyIsImF1ZCI6Imh0dHA6XC9cL2V4YW1wbGUuY29tIiwiaWF0IjoxMzU2OTk5NTI0LCJuYmYiOjEzNTcwMDAwMDAsImRhdGEiOnsiaWQiOiI5IiwiZmlyc3RuYW1lIjoiTWlrZSIsImxhc3RuYW1lIjoiRGFsaXNheSIsImVtYWlsIjoibWlrZUBjb2Rlb2ZhbmluamEuY29tIn19.h\_Q4gJ3epcpwdwNCNCYxtiKdXsN34W9MEjxZ7sx21Vs |

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

|  |
| --- |
| header.payload.signature |

The header component contains information about how the JWT signature should be computed. The payload component is the data that is stored inside the JWT. This can be the 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.

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).

**JWT vs OAuth**

We explained the JWT above. JWT is a token format and we can say it is a simple authorization 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 a 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 authorization 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).

**Final output**

**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.

[espro-slider id=19422]

**LEVEL 2 source code output**

*Screenshots coming soon! See the list of features in 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!

**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 the JWT library developed by Google.  
├────── objects/  
├───────── user.php – the class file that will handle the database queries.  
├────── create\_user.php – the file that will process the input of from “sign up” form.  
├────── login.php – the file that will encode and generate a JSON web token.  
├────── update\_user.php – the file that will process the input of from “user account” form.  
├────── validate\_token.php – the 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.

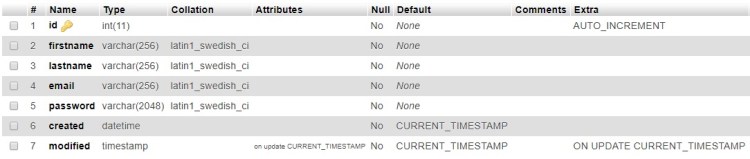
**Setup the Database**

**Create a database**

* Open your PhpMyAdmin [localhost/phpmyadmin](http://localhost/phpmyadmin/)
* Create a database called api\_db

**Create a table**

* On the api\_db database, create a new table called users.
* Put the following fields on the users table.



**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.

**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 connection  **class** Database{        // specify your own database credentials  **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("mysql:host=" . $this->host . ";dbname=" . $this->db\_name, $this->username, $this->password);          }**catch**(PDOException $exception){              echo "Connection error: " . $exception->getMessage();          }    **return** $this->conn;      }  }  ?> |

**Create API for user registration**

**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 from a specific URL. Place the following code.

|  |
| --- |
| <?php  // required headers  header("Access-Control-Allow-Origin: <http://localhost/rest-api-authentication-example/>");  header("Content-Type: application/json; charset=UTF-8");  header("Access-Control-Allow-Methods: POST");  header("Access-Control-Max-Age: 3600");  header("Access-Control-Allow-Headers: Content-Type, Access-Control-Allow-Headers, Authorization, X-Requested-With");    // database connection will be here |

**Connect to database and user table**

We are saving the user registration 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 |

**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("php://input"));    // 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 |

**Use the create() method**

One the code below, we check if the user data is not empty. We also 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 created      echo json\_encode(**array**("message" => "User was created."));  }    // message if unable to create user  **else**{        // set response code      http\_response\_code(400);        // display message: unable to create user      echo json\_encode(**array**("message" => "Unable to create user."));  }  ?> |

**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 name  **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  } |

**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->table\_name . "              SET                  firstname = :firstname,                  lastname = :lastname,                  email = :email,                  password = :password";        // prepare the query      $stmt = $this->conn->prepare($query);        // sanitize      $this->firstname=htmlspecialchars(strip\_tags($this->firstname));      $this->lastname=htmlspecialchars(strip\_tags($this->lastname));      $this->email=htmlspecialchars(strip\_tags($this->email));      $this->password=htmlspecialchars(strip\_tags($this->password));        // bind the values      $stmt->bindParam(':firstname', $this->firstname);      $stmt->bindParam(':lastname', $this->lastname);      $stmt->bindParam(':email', $this->email);        // hash the password before saving to database      $password\_hash = password\_hash($this->password, PASSWORD\_BCRYPT);      $stmt->bindParam(':password', $password\_hash);        // execute the query, also check if query was successful  **if**($stmt->execute()){  **return** true;      }    **return** false;  }    // emailExists() method will be here |

**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 the successful creation of a user**.
* Launch POSTMAN.
* Enter the following as the request URL

|  |
| --- |
| <http://localhost/rest-api-authentication-example/api/create_user.php> |

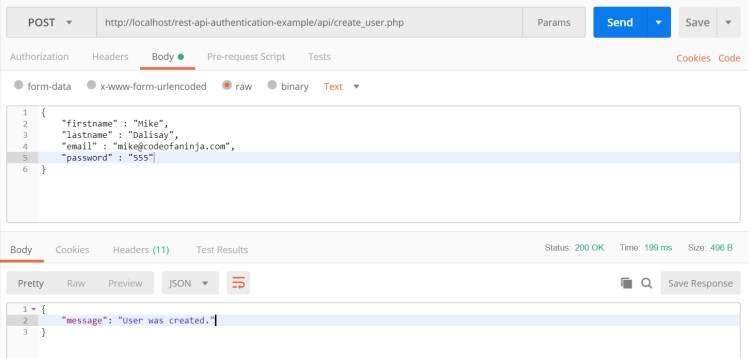
* Click “Body” tab.
* Click “raw”.
* Enter the following JSON.

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

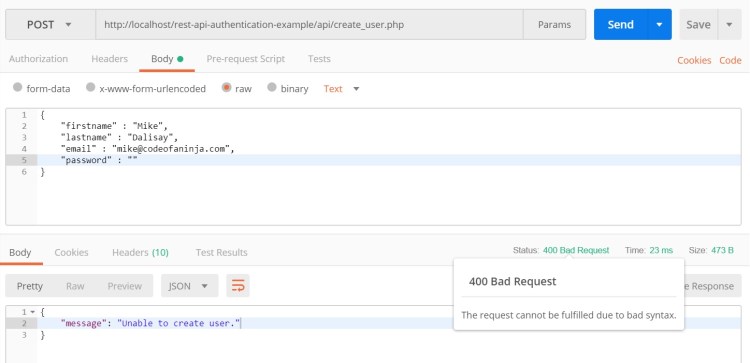
* Click the blue “Send” button. The output will be:

|  |
| --- |
| {      "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.
* Click the blue “Send” button.
* It should look like this:



**Create API for user login**

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

|  |
| --- |
| <?php  // required headers  header("Access-Control-Allow-Origin: <http://localhost/rest-api-authentication-example/>");  header("Content-Type: application/json; charset=UTF-8");  header("Access-Control-Allow-Methods: POST");  header("Access-Control-Max-Age: 3600");  header("Access-Control-Allow-Headers: Content-Type, Access-Control-Allow-Headers, Authorization, X-Requested-With");    // database connection will be here |

**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 |

**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("php://input"));    // set product property values  $user->email = $data->email;  $email\_exists = $user->emailExists();    // files for jwt will be here |

**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 database  **function** emailExists(){        // query to check if email exists      $query = "SELECT id, firstname, lastname, password              FROM " . $this->table\_name . "              WHERE email = ?              LIMIT 0,1";        // prepare the query      $stmt = $this->conn->prepare( $query );        // sanitize      $this->email=htmlspecialchars(strip\_tags($this->email));        // bind given email value      $stmt->bindParam(1, $this->email);        // execute the query      $stmt->execute();        // get number of rows      $num = $stmt->rowCount();        // if email exists, assign values to object properties for easy access and use for php sessions  **if**($num>0){            // get record details / values          $row = $stmt->fetch(PDO::FETCH\_ASSOC);            // assign values to object properties          $this->id = $row['id'];          $this->firstname = $row['firstname'];          $this->lastname = $row['lastname'];          $this->password = $row['password'];            // return true because email exists in the database  **return** true;      }        // return false if email does not exist in the database  **return** false;  }    // update() method will be here |

**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/BeforeValidException.php';  **include\_once** 'libs/php-jwt-master/src/ExpiredException.php';  **include\_once** 'libs/php-jwt-master/src/SignatureInvalidException.php';  **include\_once** 'libs/php-jwt-master/src/JWT.php';  **use** \Firebase\JWT\JWT;    // generate jwt will be here |

**Generate JSON web token**

The code below will check if the email exists and if the password matches 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 password is correct  **if**($email\_exists && password\_verify($data->password, $user->password)){        $token = **array**(         "iat" => $issued\_at,         "exp" => $expiration\_time,         "iss" => $issuer,         "data" => **array**(             "id" => $user->id,             "firstname" => $user->firstname,             "lastname" => $user->lastname,             "email" => $user->email         )      );        // set response code      http\_response\_code(200);        // generate jwt      $jwt = JWT::encode($token, $key);      echo json\_encode(  **array**(                  "message" => "Successful login.",                  "jwt" => $jwt              )          );    }    // login failed will be here |

**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" => "Login failed."));  }  ?> |

**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";  $issued\_at = time();  $expiration\_time = $issued\_at + (60 \* 60); // valid for 1 hour  $issuer = "<http://localhost/CodeOfaNinja/RestApiAuthLevel1/>";  ?> |

**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.

**Output**

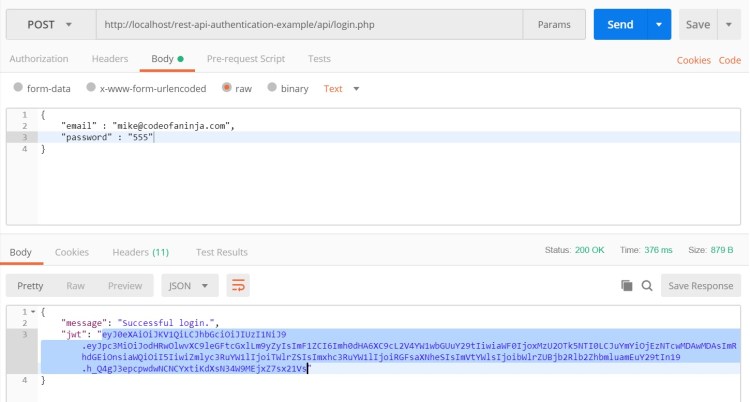
* To **test for successful login**, enter the following as the request URL.

|  |
| --- |
| <http://localhost/rest-api-authentication-example/api/login.php> |

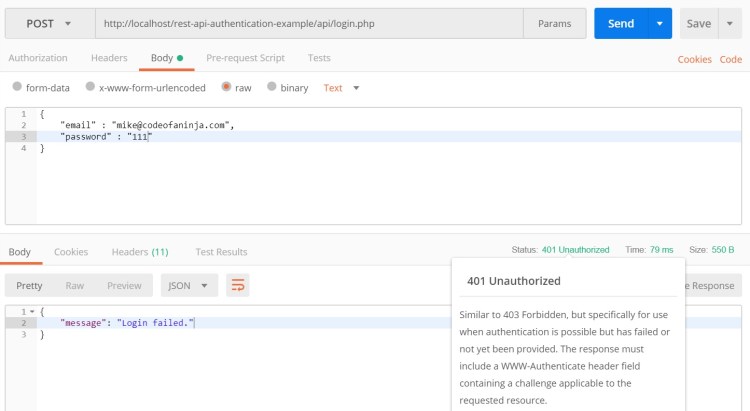
* Enter the following on the body.

|  |
| --- |
| {   "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 the wrong password.



**Create API for JWT validation**

**Create a file to validate JWT**

This file will return output in JSON format and will accept requests from the 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: <http://localhost/rest-api-authentication-example/>");  header("Content-Type: application/json; charset=UTF-8");  header("Access-Control-Allow-Methods: POST");  header("Access-Control-Max-Age: 3600");  header("Access-Control-Allow-Headers: Content-Type, Access-Control-Allow-Headers, Authorization, X-Requested-With");    // files for decoding jwt will be here |

**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/BeforeValidException.php';  **include\_once** 'libs/php-jwt-master/src/ExpiredException.php';  **include\_once** 'libs/php-jwt-master/src/SignatureInvalidException.php';  **include\_once** 'libs/php-jwt-master/src/JWT.php';  **use** \Firebase\JWT\JWT;    // retrieve gieve jwt here |

**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("php://input"));    // get jwt  $jwt=isset($data->jwt) ? $data->jwt : "";    // decode jwt here |

**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 details  **try** {          // decode jwt          $decoded = JWT::decode($jwt, $key, **array**('HS256'));            // set response code          http\_response\_code(200);            // show user details          echo json\_encode(**array**(              "message" => "Access granted.",              "data" => $decoded->data          ));        }        // catch will be here  }    // error if jwt is empty will be here |

**Show error if decoding failed**

If decoding JWT failed, it means 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 invalid  **catch** (Exception $e){        // set response code      http\_response\_code(401);        // tell the user access denied  & show error message      echo json\_encode(**array**(          "message" => "Access denied.",          "error" => $e->getMessage()      ));  } |

**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" => "Access denied."));  }  ?> |

**Output**

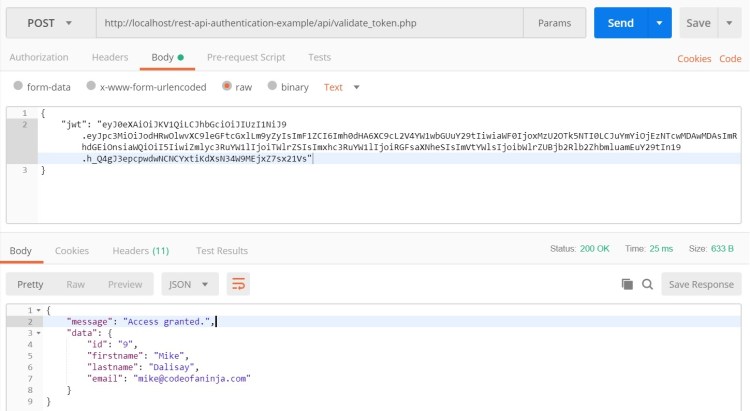
* To **test for successful access**, enter the following request URL.

|  |
| --- |
| <http://localhost/rest-api-authentication-example/api/validate_token.php> |

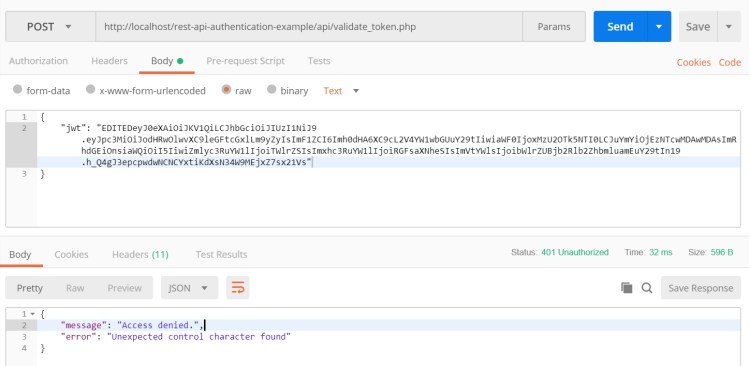
* Enter the JSON Web Token we retrieved earlier. The JSON web token below is different from yours. Make sure your JWT was generated in your machine.

|  |
| --- |
| {      "jwt": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJodHRwOlwvXC9leGFtcGxlLm9yZyIsImF1ZCI6Imh0dHA6XC9cL2V4YW1wbGUuY29tIiwiaWF0IjoxMzU2OTk5NTI0LCJuYmYiOjEzNTcwMDAwMDAsImRhdGEiOnsiaWQiOiI5IiwiZmlyc3RuYW1lIjoiTWlrZSIsImxhc3RuYW1lIjoiRGFsaXNheSIsImVtYWlsIjoibWlrZUBjb2Rlb2ZhbmluamEuY29tIn19.h\_Q4gJ3epcpwdwNCNCYxtiKdXsN34W9MEjxZ7sx21Vs"  } |

* It should look like this on POSTMAN.



* To **test for failed access**, just put the word “EDITED” on your JWT. This will make JWT wrong. It will result in denied access.
* It should look like the following.



**Create API for user account**

**Create a file for updating user account**

This file will return output in JSON format and will accept requests from the 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; charset=UTF-8");  header("Access-Control-Allow-Methods: POST");  header("Access-Control-Max-Age: 3600");  header("Access-Control-Allow-Headers: Content-Type, Access-Control-Allow-Headers, Authorization, X-Requested-With");    // files for decoding jwt will be here |

**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/BeforeValidException.php';  **include\_once** 'libs/php-jwt-master/src/ExpiredException.php';  **include\_once** 'libs/php-jwt-master/src/SignatureInvalidException.php';  **include\_once** 'libs/php-jwt-master/src/JWT.php';  **use** \Firebase\JWT\JWT;    // database connection will be here |

**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 |

**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("php://input"));    // get jwt  $jwt=isset($data->jwt) ? $data->jwt : "";    // decode jwt here |

**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 details  **try** {            // decode jwt          $decoded = JWT::decode($jwt, $key, **array**('HS256'));            // set user property values here      }        // catch failed decoding will be here  }    // error message if jwt is empty will be here |

**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 invalid  **catch** (Exception $e){        // set response code      http\_response\_code(401);        // show error message      echo json\_encode(**array**(          "message" => "Access denied.",          "error" => $e->getMessage()      ));  } |

**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 |

**Use the update() method**

One the code below, we use the user object’s create() method. If it returns true, it means the 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" => "Unable to update user."));  } |

**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->password) ? ", password = :password" : "";        // if no posted password, do not update the password      $query = "UPDATE " . $this->table\_name . "              SET                  firstname = :firstname,                  lastname = :lastname,                  email = :email                  {$password\_set}              WHERE id = :id";        // prepare the query      $stmt = $this->conn->prepare($query);        // sanitize      $this->firstname=htmlspecialchars(strip\_tags($this->firstname));      $this->lastname=htmlspecialchars(strip\_tags($this->lastname));      $this->email=htmlspecialchars(strip\_tags($this->email));        // bind the values from the form      $stmt->bindParam(':firstname', $this->firstname);      $stmt->bindParam(':lastname', $this->lastname);      $stmt->bindParam(':email', $this->email);        // hash the password before saving to database  **if**(!empty($this->password)){          $this->password=htmlspecialchars(strip\_tags($this->password));          $password\_hash = password\_hash($this->password, PASSWORD\_BCRYPT);          $stmt->bindParam(':password', $password\_hash);      }        // unique ID of record to be edited      $stmt->bindParam(':id', $this->id);        // execute the query  **if**($stmt->execute()){  **return** true;      }    **return** false;  } |

**Re-generate JWT**

We need to re-generate or get a new JSON Web Token especially if user information was changed. The code below does that and it sets a response code of 200 and tells 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 user details might be different  $token = **array**(     "iat" => $issued\_at,     "exp" => $expiration\_time,     "iss" => $issuer,     "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 updated.",              "jwt" => $jwt          )      ); |

**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" => "Access denied."));  }  ?> |

**Output**

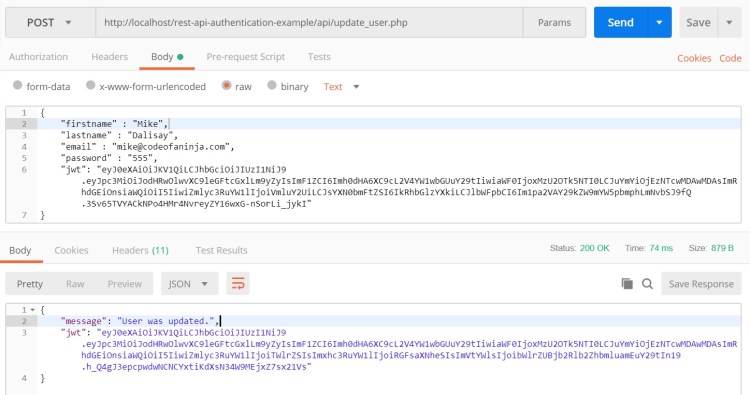
* To **test for successful user update**, enter the following as request URL on POSTMAN.

|  |
| --- |
| <http://localhost/rest-api-authentication-example/api/update_user.php> |

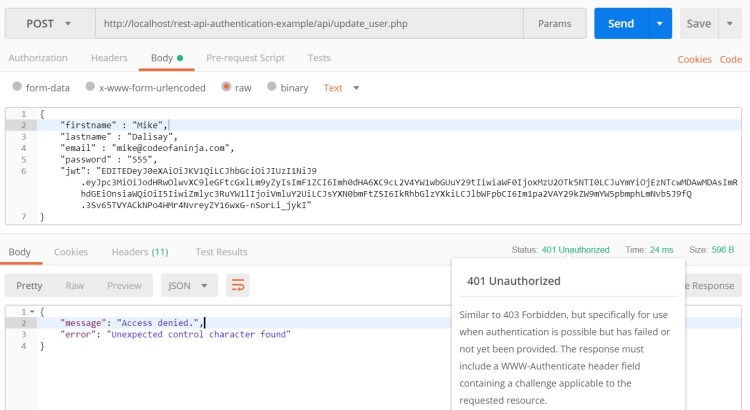
* On the body section, enter new user information with the JSON Web Token we retrieved earlier.
* The JSON web token below is different from yours. Make sure your JWT was generated in your machine.

|  |
| --- |
| {      "firstname" : "Mike",      "lastname" : "Dalisay",      "email" : "mike@codeofaninja.com",      "password" : "555",      "jwt": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJodHRwOlwvXC9leGFtcGxlLm9yZyIsImF1ZCI6Imh0dHA6XC9cL2V4YW1wbGUuY29tIiwiaWF0IjoxMzU2OTk5NTI0LCJuYmYiOjEzNTcwMDAwMDAsImRhdGEiOnsiaWQiOiI5IiwiZmlyc3RuYW1lIjoiVmluY2UiLCJsYXN0bmFtZSI6IkRhbGlzYXkiLCJlbWFpbCI6Im1pa2VAY29kZW9mYW5pbmphLmNvbSJ9fQ.3Sv65TVYACkNPo4HMr4NvreyZY16wxG-nSorLi\_jykI"  } |

* It should look like this on POSTMAN.



* As you can see in 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)

**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="width=device-width, initial-scale=1, shrink-to-fit=no" />            <**title**>Rest API Authentication Example</**title**>            <!-- CSS links will be here -->        </**head**>  <**body**>    <!-- navigation bar will be here -->    <!-- script links will be here -->    </**body**>  </**html**> |

**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 navbar-dark bg-dark fixed-top">      <**a** class="navbar-brand" href="#">Navbar</**a**>      <**button** class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNavAltMarkup" aria-controls="navbarNavAltMarkup" aria-expanded="false" aria-label="Toggle navigation">          <**span** class="navbar-toggler-icon"></**span**>      </**button**>      <**div** class="collapse navbar-collapse" id="navbarNavAltMarkup">          <**div** class="navbar-nav">              <**a** class="nav-item nav-link" href="#" id='home'>Home</**a**>              <**a** class="nav-item nav-link" href="#" id='update\_account'>Account</**a**>              <**a** class="nav-item nav-link" href="#" id='logout'>Logout</**a**>              <**a** class="nav-item nav-link" href="#" id='login'>Login</**a**>              <**a** class="nav-item nav-link" href="#" id='sign\_up'>Sign Up</**a**>          </**div**>      </**div**>  </**nav**>  <!-- /navbar -->    <!-- content section will be here --> |

**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 starter-template">        <**div** class="row">          <**div** class="col">                <!-- where prompt / messages will appear -->              <**div** id="response"></**div**>                <!-- where main content will appear -->              <**div** id="content"></**div**>          </**div**>      </**div**>    </**main**>  <!-- /container --> |

**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://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css>" integrity="sha384-Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm" crossorigin="anonymous" />  <**link** rel="stylesheet" type="text/css" href="custom.css" /> |

**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 1.5rem; }  #logout{ **display**:none; } |

**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 libraries -->  <**script** src="<https://code.jquery.com/jquery-3.2.1.min.js>"></**script**>  <**script** src="<https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.12.9/umd/popper.min.js>" integrity="sha384-ApNbgh9B+Y1QKtv3Rn7W3mgPxhU9K/ScQsAP7hUibX39j7fakFPskvXusvfa0b4Q" crossorigin="anonymous"></**script**>  <**script** src="<https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootstrap.min.js>" integrity="sha384-JZR6Spejh4U02d8jOt6vLEHfe/JQGiRRSQQxSfFWpi1MquVdAyjUar5+76PVCmYl" crossorigin="anonymous"></**script**>    <!-- jquery scripts will be here --> |

**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', **function**(){    **var** html = `              <h2>Sign Up</h2>              <form id='sign\_up\_form'>                  <div **class**="form-group">                      <label **for**="firstname">Firstname</label>                      <input type="text" **class**="form-control" name="firstname" id="firstname" required />                  </div>                    <div **class**="form-group">                      <label **for**="lastname">Lastname</label>                      <input type="text" **class**="form-control" name="lastname" id="lastname" required />                  </div>                    <div **class**="form-group">                      <label **for**="email">Email</label>                      <input type="email" **class**="form-control" name="email" id="email" required />                  </div>                    <div **class**="form-group">                      <label **for**="password">Password</label>                      <input type="password" **class**="form-control" name="password" id="password" required />                  </div>                    <button type='submit' **class**='btn btn-primary'>Sign Up</button>              </form>              `;            clearResponse();          $('#content').html(html);      });        // trigger when registration form is submitted here        // show login form trigger will be here        // clearResponse() will be here  });  </script> |

**Trigger when sign up form is submitted**

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

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

|  |
| --- |
| // trigger when registration form is submitted  $(document).on('submit', '#sign\_up\_form', **function**(){        // get form data  **var** sign\_up\_form=$(**this**);  **var** form\_data=JSON.stringify(sign\_up\_form.serializeObject());        // submit form data to api      $.ajax({          url: "api/create\_user.php",          type : "POST",          contentType : 'application/json',          data : form\_data,          success : **function**(result) {              // if response is a success, tell the user it was a successful sign up & empty the input boxes              $('#response').html("<div class='alert alert-success'>Successful sign up. Please login.</div>");              sign\_up\_form.find('input').val('');          },          error: **function**(xhr, resp, text){              // on error, tell the user sign up failed              $('#response').html("<div class='alert alert-danger'>Unable to sign up. Please contact admin.</div>");          }      });    **return** **false**;  }); |

**Remove any prompt messages**

The clearResponse() method was used in 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 |

**Add serializeObject function**

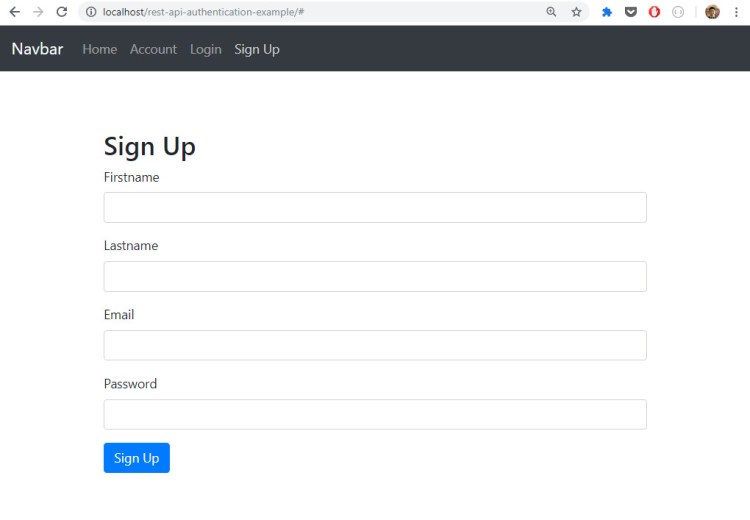
The serializeObject function will convert form data to JSON format. We need this function to send values from an HTML form to the API.

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

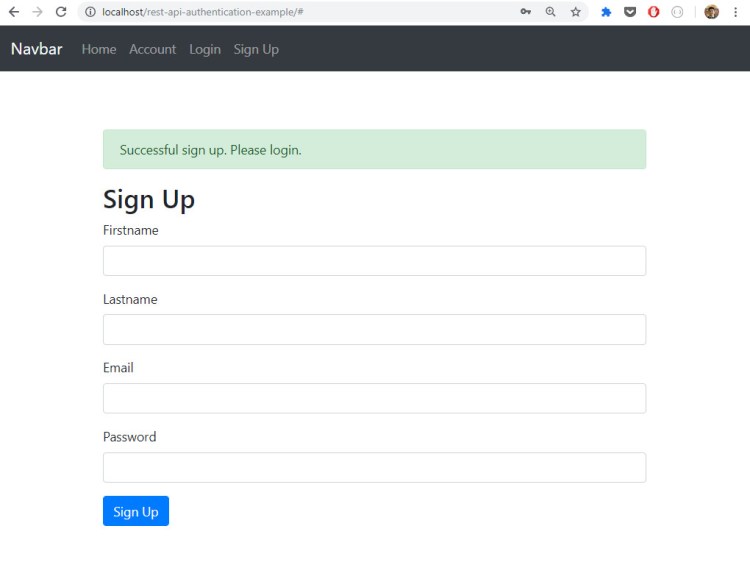
|  |
| --- |
| // function to make form values to json format  $.fn.serializeObject = **function**(){    **var** o = {};  **var** a = **this**.serializeArray();      $.each(a, **function**() {  **if** (o[**this**.name] !== undefined) {  **if** (!o[**this**.name].push) {                  o[**this**.name] = [o[**this**.name]];              }              o[**this**.name].push(**this**.value || '');          } **else** {              o[**this**.name] = **this**.value || '';          }      });  **return** o;  }; |

**Output**

When the user clicks the Sign-Up link on the navigation bar.

[](https://i0.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://i0.wp.com/www.codeofaninja.com/wp-content/uploads/2018/09/4-successful-sign-up.jpg?ssl=1)

**Create a login page**

**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', **function**(){      showLoginPage();  });    // login form submit trigger will be here |

**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'>Email address</label>                  <input type='email' **class**='form-control' id='email' name='email' placeholder='Enter email'>              </div>                <div **class**='form-group'>                  <label **for**='password'>Password</label>                  <input type='password' **class**='form-control' id='password' name='password' placeholder='Password'>              </div>                <button type='submit' **class**='btn btn-primary'>Login</button>          </form>          `;        $('#content').html(html);      clearResponse();      showLoggedOutMenu();  }    // setCookie() will be here    // showLoggedOutMenu() will be here |

**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, exdays) {  **var** d = **new** Date();      d.setTime(d.getTime() + (exdays\*24\*60\*60\*1000));  **var** expires = "expires="+ d.toUTCString();      document.cookie = cname + "=" + cvalue + ";" + expires + ";path=/";  } |

**Change menu appearance**

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

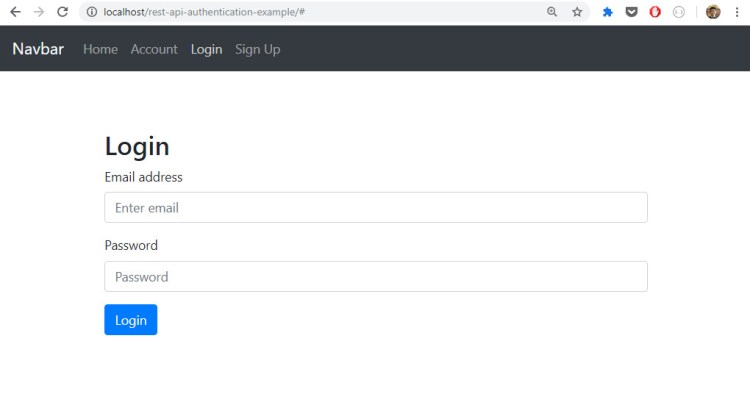
This function will make the menu look like the options for a logged-out user.

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 navbar & hide logout button      $("#login, #sign\_up").show();      $("#logout").hide();  }    // showHomePage() function will be here |

**Output**

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

[](https://i0.wp.com/www.codeofaninja.com/wp-content/uploads/2018/09/5-login.jpg?ssl=1)

**Show login responses**

**Submitted form trigger**

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

It gets the data from the form and stores 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  $(document).on('submit', '#login\_form', **function**(){        // get form data  **var** login\_form=$(**this**);  **var** form\_data=JSON.stringify(login\_form.serializeObject());        // http request will be here    **return** **false**;  });    // trigger to show home page will be here |

**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 are 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 user it was a successful login          showHomePage();          $('#response').html("<div class='alert alert-success'>Successful login.</div>");        },      // error response will be here  }); |

**Show home page HTML**

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", JSON.stringify({ jwt:jwt })).done(**function**(result) {            // home page html will be here      })        // show login page on error will be here  }    // getCookie() will be here    // showLoggedInMenu() will be here |

**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">Welcome to Home!</div>          <div **class**="card-body">              <h5 **class**="card-title">You are logged **in**.</h5>              <p **class**="card-text">You won't be able to access the home and account pages if you are not logged in.</p>          </div>      </div>      `;    $('#content').html(html);  showLoggedInMenu(); |

**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 = decodeURIComponent(document.cookie);  **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.length, c.length);          }      }  **return** "";  } |

**Show error for invalid login**

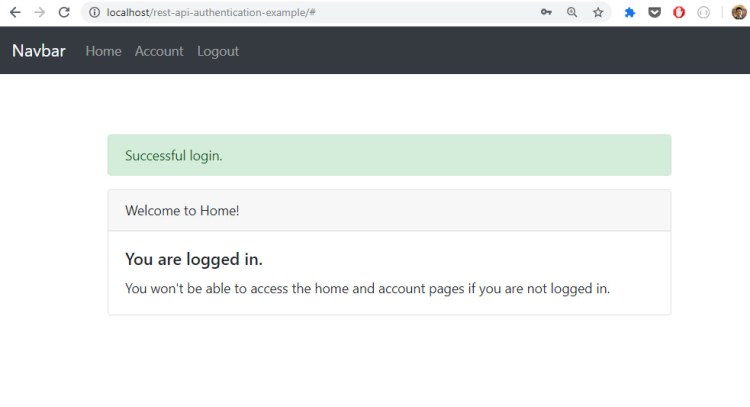
If the submitted email and password are invalid, we tell the user login failed and empty the login form.

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

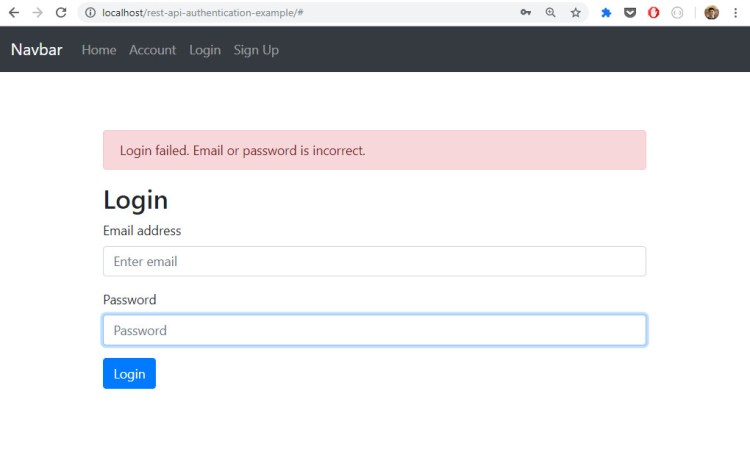
|  |
| --- |
| error: **function**(xhr, resp, text){      // on error, tell the user login has failed & empty the input boxes      $('#response').html("<div class='alert alert-danger'>Login failed. Email or password is incorrect.</div>");      login\_form.find('input').val('');  } |

**Output**

* If the user entered a valid email and password.

[](https://i0.wp.com/www.codeofaninja.com/wp-content/uploads/2018/09/6-successful-login-screen.jpg?ssl=1)

* If the user entered an invalid email or password.

[](https://i0.wp.com/www.codeofaninja.com/wp-content/uploads/2018/09/7-failed-login-screen.jpg?ssl=1)

**On click of Home menu**

**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', **function**(){      showHomePage();      clearResponse();  });    // trigger to show account form will be here |

**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 navbar & show logout button      $("#login, #sign\_up").hide();      $("#logout").show();  }    // showUpdateAccountForm() will be here |

**Show login page if JWT is invalid**

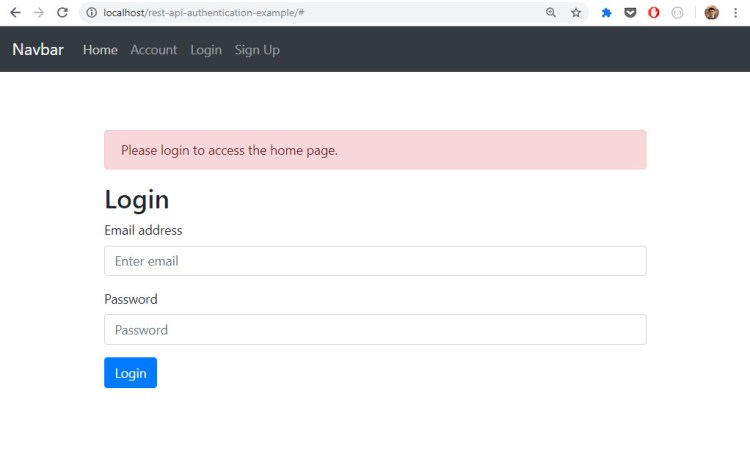
If JWT is invalid, we will show the login page and ask the user to login.

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

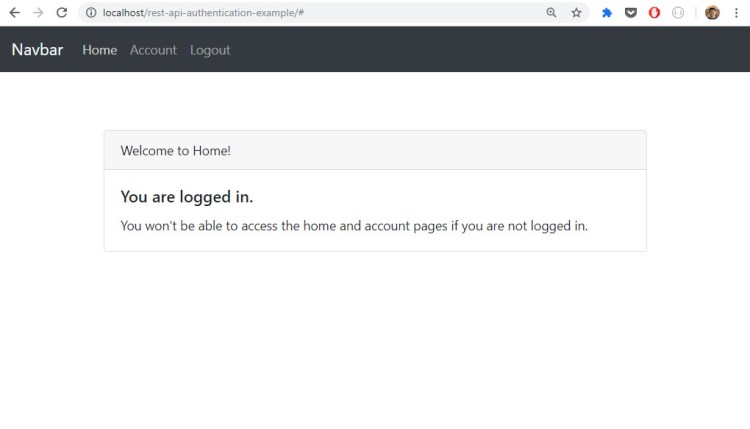
|  |
| --- |
| // show login page on error  .fail(**function**(result){      showLoginPage();      $('#response').html("<div class='alert alert-danger'>Please login to access the home page.</div>");  }); |

**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)

**Show user account page**

**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\_account', **function**(){      showUpdateAccountForm();  });    // trigger for updating user account will be here |

**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", JSON.stringify({ jwt:jwt })).done(**function**(result) {            // html form for updating user account will be here      })        // error message when jwt is invalid will be here  } |

**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 details in the form  **var** html = `          <h2>Update Account</h2>          <form id='update\_account\_form'>              <div **class**="form-group">                  <label **for**="firstname">Firstname</label>                  <input type="text" **class**="form-control" name="firstname" id="firstname" required value="` + result.data.firstname + `" />              </div>                <div **class**="form-group">                  <label **for**="lastname">Lastname</label>                  <input type="text" **class**="form-control" name="lastname" id="lastname" required value="` + result.data.lastname + `" />              </div>                <div **class**="form-group">                  <label **for**="email">Email</label>                  <input type="email" **class**="form-control" name="email" id="email" required value="` + result.data.email + `" />              </div>                <div **class**="form-group">                  <label **for**="password">Password</label>                  <input type="password" **class**="form-control" name="password" id="password" />              </div>                <button type='submit' **class**='btn btn-primary'>                  Save Changes              </button>          </form>      `;    clearResponse();  $('#content').html(html); |

**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 needs to login to show the account page  .fail(**function**(result){      showLoginPage();      $('#response').html("<div class='alert alert-danger'>Please login to access the account page.</div>");  }); |

**Add a trigger for updating a 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 is submitted  $(document).on('submit', '#update\_account\_form', **function**(){        // handle for update\_account\_form  **var** update\_account\_form=$(**this**);        // validate jwt to verify access  **var** jwt = getCookie('jwt');        // get form data and jwt here    **return** **false**;  });    // trigger to logout will be here |

**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\_account\_form.serializeObject()    // add jwt on the object  update\_account\_form\_obj.jwt = jwt;    // convert object to json string  **var** form\_data=JSON.stringify(update\_account\_form\_obj);    // send data to api here |

**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 updated          $('#response').html("<div class='alert alert-success'>Account was updated.</div>");            // store new jwt to coookie          setCookie("jwt", result.jwt, 1);      },        // errors will be handled here  }); |

**Show error message**

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

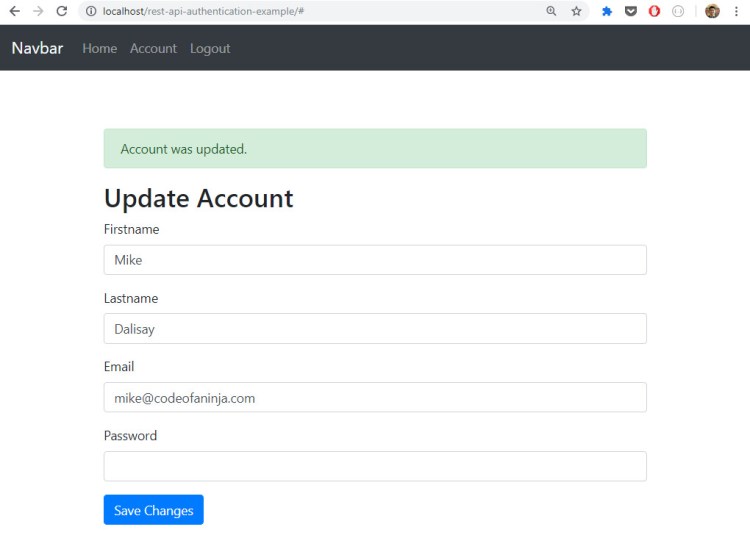
If JWT is invalid and access is denied, we logout the user and ask him to log in.

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

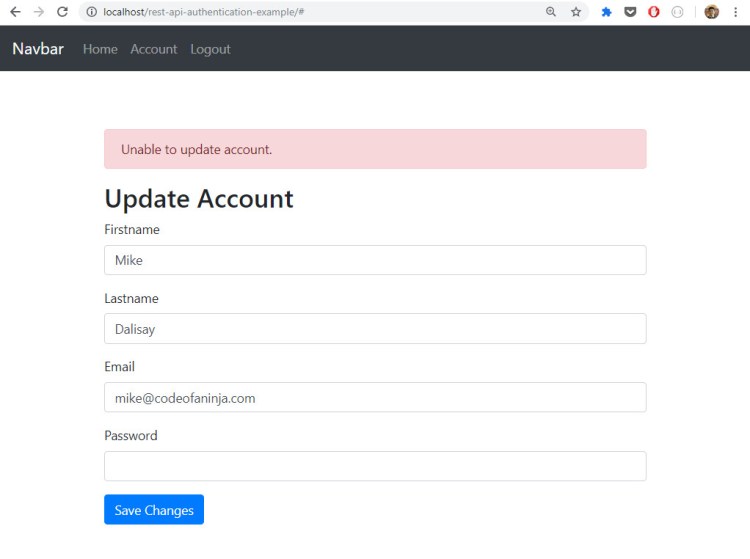
|  |
| --- |
| // show error message to user  error: **function**(xhr, resp, text){  **if**(xhr.responseJSON.message=="Unable to update user."){          $('#response').html("<div class='alert alert-danger'>Unable to update account.</div>");      }    **else** **if**(xhr.responseJSON.message=="Access denied."){          showLoginPage();          $('#response').html("<div class='alert alert-success'>Access denied. Please login</div>");      }  } |

**Output**

* If a user account was updated successfully.

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

* If there was a problem when updating a user account.

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

**Add JavaScript for user logout**

**Add a trigger to logout**

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

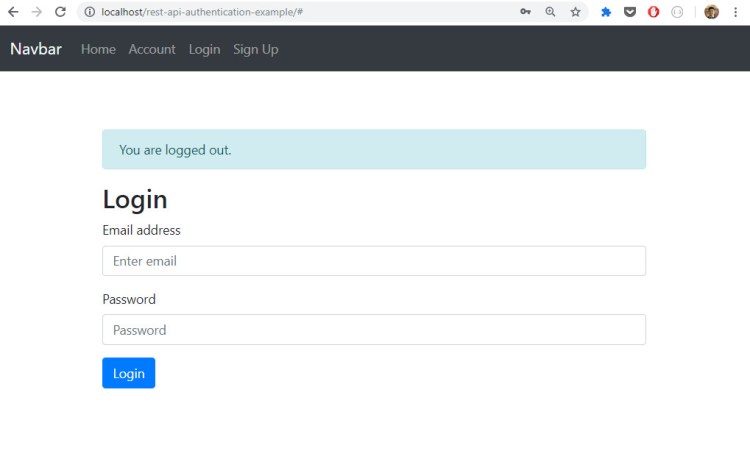
We use the showLoginPage(); method to logout the user. We tell he is logged out as well.

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

|  |
| --- |
| // logout the user  $(document).on('click', '#logout', **function**(){      showLoginPage();      $('#response').html("<div class='alert alert-info'>You are logged out.</div>");  }); |

**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)