Ecommerce Design – Restful API for an Online Store

Objective

Design a restful API for an online store which can be used to manage different products. We will create a REST API which will implement CRUD (add, view, edit, delete) operation on products table from outside the application.

Important Security Issue

For the time being I am not deleting setup.php but it should be because it has database credential.

Assumptions

- Product name is unique. It should not contain white space.
- Database should be already create
- Product table fields-id, name, description, supplier_name, created_at, updated_at
- User table fields- id, name, email, password

Implementation

Products Table Schema

#	Name	Type	Collation	Attributes	Null	Default	Extra	Action
1	id	int(11)			No	None	AUTO_INCREMENT	Change Drop Primary Unique Index More
2	name	varchar(250)			Yes	NULL		Change Drop Primary Unique Index More
3	description	text			No	None		Change Drop Primary Unique Index More
4	supplier_name	varchar(250)			Yes	NULL		Change Drop Primary Unique Index More
5	created_at	timestamp			No	CURRENT_TIMESTAMP		Change Drop Primary Unique Index More
6	updated_at	timestamp			No	CURRENT_TIMESTAMP		Change Drop Primary Unique Index More

Users Table

#	Name	Type	Collation	Attributes	Null	Default	Extra	Action
1	id	int(11)			No	None	AUTO_INCREMENT	Change Drop Primary Unique Index More
2	name	varchar(250)			Yes	NULL		Change Drop Primary Unique Index More
3	email	varchar(255)			No	None		Change Drop Primary Unique Index More
4	password_hash	text			No	None		Change Drop Primary Unique Index More
5	api_key	varchar(32)			No	None		Change Drop Primary Unique Index More
6	created_at	timestamp			No	CURRENT_TIMESTAMP		Change Drop Primary Unique Index More

id => Integer (Primary Key)

name => Varchar(100)

email => Varchar (100) (Unique Key)

password_hash => text (Not NULL)

api_key => Varchar (100) (Unique Key)

API URL Structure

URL http://localhost:3000/.....	**Method**	**Parameters**	**Description**
/register	POST	name, email, password	User registration
/login	POST	email, password	User login

/add	POST	name, description, supplier_name	Add new product
/view	GET		Fetching all product
/edit/:name	POST	name, updated values	Update single product
/delete/:name	DELETE	name	Deleting single product

HTTP Status Code

HTTP status codes in the response body tell client application what action should be taken with the response. For an example if the response code 200, it means on the server side the request is processed successfully and you can expect updated data in the response. As well if the status code is 401, the request is not authorized. An example cause for 401 could be API key is invalid.

200	OK
201	Created
202	User already exist
400	Bad Request, missing Parameter, invalid email, incorrect credential, fail to add product , fail to update, fail to delete
401	Unauthorized, API key missing
404	Path Not Found
500	Internal Server Error

API Authentication

Find below step for user authentication:

- Check authenticity is executed before every product API call.
- At user level we are storing api_key which is unique per user.
- api_key will be passed in header.

API key in the request header by reading **Authorization** field. Basically we'll look into database for matched API key and get the appropriate user. If the API key not present in users table, then we'll stop the execution and echo the error json.

The method **authenticate ()** will be executed every time before doing any task related operations on database.

If the API key is missing in the request header, the following json will be echoed With 401 status code.

I divided my tests in 3 parts

1. Authentication tests:

1. Can we login with email id and password
2. What if email id is invalid
3. What if both field are empty and front end developer not handled them
4. What if user fill malicious data intend to SQL injection, XSS

2. Unit tests: add product, edit product, delete product, view

1. Can we add product - HTTP POST
2. Can we add same products more than one time - HTTP POST
3. Can we add product with empty field
4. Can we add product when some field are empty
5. Can we add product when API key is missing
6. Can we delete a product by ID - HTTP DELETE
7. Can we show all product - HTTP GET
8. Can we update a product by ID and data - HTTP POST
9. Can we update with some field of a product
10. Can we update same name o product multiple times
11. These are again simple cases once these are done I went on to create edge test cases like

3 .Error codes tests: Upon failure are the API's returning correct error response codes.

Test Case Run Constraints

1. Product name should not contain white space.

2. Test cases for add, edit, delete product are inter related so make sure before update and delete a product you have added that product in database in add product test case.

3. During update product name make sure you are updating with name which don't exist in database because product name is unique if already exist it will not update in database.

4. Make sure during delete test case you are deleting all product which you have added either when add test case run again product name already exist in the database so test case will fail.

Testing the API

Response format

1. Register -> Upon the successful registration the following json response will be issued.

```
{
    "error": “false”,
    "message": "You are successfully registered",
}
```

Fail Registration

```
{
    "error": “true”,
    "message": "Required field(s) name, email, password is missing or empty",
}
```

User all ready exist

```
{
    "error": “true”,
    "message": " Sorry, this email already existed",
}
```

2. Login -> On successful login the following json will be issued.

```
{
  "error": false,
  "name": "Ramkesh",
  "email": "ram@gmail.com",
  "apiKey": "cbe8edfc56be6d2aaff74b186eabbb84",
  "createdAt": "2017-11-19 23:38:35"
}
```

If the credentials are wrong, you can expect the following json

```
{
  "error": true,
  "message": "Login failed. Incorrect credentials"
}
```

If email is not valid

```
{
  "error": true,
  "message": "Invalid Email"
}
```

4. Add Products

4.1 Success

```
{
  "error": false,
  "message": "Product added successfully"
}
```


4.2 Failure

If product name already exist


```
{
  "error": true,
  "message": "Failed to add product. Please try again"
}
```


If product list parameter is missing


```
{
  "error": true,
  "message": "Missing product field"
}
```


If API key is not authorize


```
{
  "error": true,
  "message": "Unauthorized"
}
```


3. Listing all product

Success


```
{
"error": false,
"products": [
{
"id": 2,
"name": "amazon",
"desciption": "amazon  web server ",
"supplier_name": "amazon",
"created_at": "2016-11-19 13:41:35",
"updated_at": "2016-11-19 13:41:35"
}
],
}
```


Fail : API key is missing


```
{
   "error": true,
   "message": "Unauthorized"
}
```


5. Updating Products

5.1 Success


```
{
   "error": false,
   "message": "Product updated successfully"
}
```

5.2 Fail – Product name not found or updated name already exist


```
{
   "error": true,
   "message": "Failed to updated product. Please try again"
}
```


5.3 Fail : API key is missing


```
{
   "error": true,
   "message": "Unauthorized"
}
```


6. Deleting Product

6.1 Success


```
   {
   "error": false,
   "message": "Product deleted successfully"
  }
```


6.2 Fail - product name not exist or internal error


```
{
   "error": true,
   "message": "Failed to delete product. Please try again"
}
```


6.3 Fail: API key is missing


```
{
   "error": true,
   "message": "Unauthorized"
}
```