INTERNATIONAL JOURNAL OF ADVANCE RESEARCH, IDEAS AND INNOVATIONS IN TECHNOLOGY

ISSN: 2454-132X
Impact factor: 6.078
(Volume 6, Issue 2)
Available online at: www.ijariit.com

E-Commerce REST API with JWT authentication

Mariam Ahmedullah
ahmedullahmaryam@gmail.com
ISL Engineering College, Hyderabad, Telangana

ABSTRACT

E-commerce is one of the fastest-growing industries around the world. Therefore, it is of immense importance to build these services with a reliable backend coupled with high-grade security. The study of this project is to integrate REST API services into the production of e-commerce backend coupled with JWT authentication system

Keywords— *E-commerce, REST API, JWT authentication*

1. INTRODUCTION

E-commerce is fast growing as an accepted and used business-standard. More and more businesses are implementing websites for performing commercial transactions over the internet. It's reasonable to mention that the method of shopping online is becoming conventional [1]. For example, the credit card company Visa reported its clients' Internet purchases to be $13 billion mark in the year 2000. This figure was expected to reach the $100 billion mark by the year 2003 [2].

Proving true, the overall e-commerce sales has been growing more than 25% and Amazon alone reported a 28% sales increase year-on-year in the year 2003. At the end of 2015, the e-commerce company announced the increase in active customers to 304 million. As of 2019, the company has 150.6 million users in the United States alone [3].

The statistics state the ever-rising growth of the industry. Hence, it is of utmost importance to create a website that is fast, asynchronous as well as secure. REST API's provide precisely that. Restful APIs and APIs have become an essential tool for backend development. Without APIs, there'll be tons of security issues within the system. The Modern Mobile and Web Development world essential rely on Restful APIs and APIs for authentication and Submitting Data. [4]

2. NODEJS

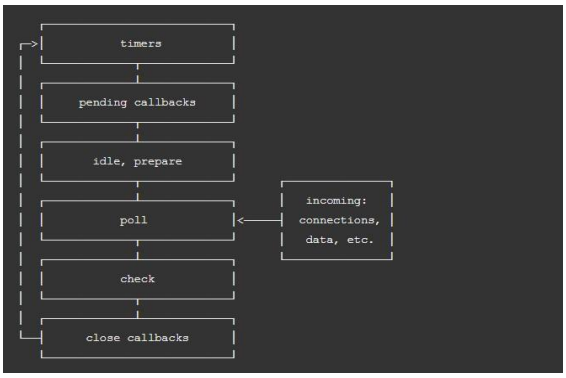
"Node.js is an open-source, cross-platform, JavaScript runtime environment that executes JavaScript code outside of a browser" [6]. It means that Nodejs allows us to run JavaScript at the server-side and helps in making powerful servers that are fast and highly scalable. This a huge plus point as many developers already know a bit of JavaScript.

2.1 Blocking vs Non-blocking

Nodejs is an event-driven single-threaded as well as non-blocking I/O model[6].It means that the processes run on a single thread as opposed to assigning a thread for each process. these processes are of two types: synchronous and asynchronous. Here, the meaning of asynchronous is that the processes can be executed behind the code without blocking/halting the entire code. Hence, the name non-blocking. This makes the code efficient and fast without crashing it. synchronous processes block the entire code while it is being executed,.this is not ideal but can be used in many situations like reading a file that is read only once. To make our code asynchronous, callback functions are used. But as this becomes important to note that having callbacks in the code does not make it asynchronous. The con of using callbacks is that it can very quickly lead to what is called a "Callback hell", which is a lot of callbacks inside a callback. This can create debugging and efficiency problems. the solution for this is to implement promises or use async await. the errors are handled in try-catch block.

2.2 Event loop

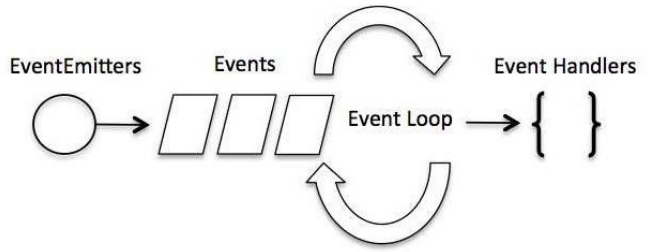
Node.js performs non-blocking I/O operations by offloading heavy operations to the thread pool whenever possible is only helped by Event Loop [8]. The event loop does the orchestration of the code. This event loop is used in the event driven architecture of nodejs. Observer pattern is used by nodejs. Event loop is kept by node thread and whenever an event is emitted, event loop picks it up and callback functions are called to execute [9]. Some of the operations performed in the event loop are depicted in the below diagram:



Source: nodejs.org

2.3 Event driven architecture

The efficiency of Nodejs is offered by Event driven architecture. As the node starts, variables are declared, functions are called and then it has to wait for the occurrence of the event, a main loop is generally present which listens for events, and the callback functions are called which are attached to the event [9].



Source: tutorialspoint

2.4 Express Framework

Express, is a web application framework for Node.js, It is intended for building web applications and APIs. It has been termed the standard server framework for Node.js [6].it can be viewed as a layer on top of Nodejs that provides better abstraction. Express contains a very strong set of features: efficient routing, easier handling of requests and responses, middlewares, server-side rendering, etc. Express helps to organize our code into the MVC architecture [10].it makes use of middleware that runs between the request-response cycles. Hence, using Nodejs and express with to build our rest API is perfect.

2.5 MongoDB with Mongoose

MongoDB is a schema-less NoSQL document database. As a NoSQL database, it helps in quick application development as well as makes it easy to deploy [12]. The data is stored in documents and these documents are stored in collections. The data is stored in BSON format that is JSON but typed.

Mongoose is an Object Data Modelling (ODM) library for MongoDB and Node.js; it provides a better level of abstraction for the database. A Mongoose model may be a wrapper on the Mongoose schema. A Mongoose schema helps us in modelling our data by defining default values, validators, etc, and Mongoose model provides an interface to the database to create documents based on the schema and perform various CRUD operations [12].

3. REST API

RE presentational State Transfer application interface. Is an architectural style including some constraints that must be completed to develop web services. REST has 6 guiding constraints which must be satisfied if an interface needs to be referred to as RESTful. These principles are listed below.

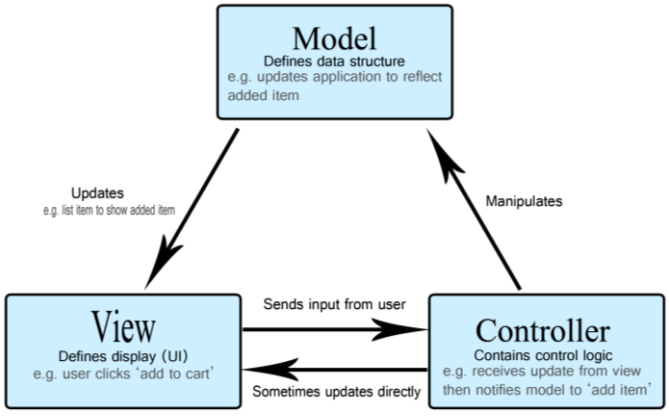
- **Client–sserver:** User interface and data storage are separated to provide a better interface and make the backend scalable as well as less complex
- **Stateless:** the requests from client-server must have all the necessary data.it must not use any stored data. The request must be stateless**.**
- **Cacheable:** Caching of data and responses must be done to increase performance and reduce the load.
- **Layered system:** A client cannot tell whether it's connected on to the top server. Layers may improve system scalability by load-balancing. Layers can also enforce security policies [11].
- **Uniform interface:** This constraint specifies the interface between clients and servers. It simplifies and decouples the architecture, which enables each part to evolve independently [11] for us, this means:
  - HTTP Verbs (GET, POST, PUT, DELETE)
  - URLs (resource name)
  - HTTP response (status and body**)**
- **Code on demand (optional):** This condition allows the customer to run some code on demand, that is, extend part of server logic to the client, either through an applet or scripts. As this item is not part of the architecture itself, it is considered optional [11].

3.1 Creating REST API with MongoDB

- After the installation of all the npm packages (nodejs, express, mongodb & mongoose), the server is created on the localhost and the database is connected.
- Models for the resources are created. For example, the product model. The fields that are important are marked as required and all the data types of the fields are specified.
- The above created models allow use to perform all the CRUD operations on our model.
  - Create: This allows us to create new documents based on the mongoose schema defined in the model. Anything that is not specified in the schema is discarded and a collection is created. here, the products collection is created in our database.
  - Read: This allows us to read or get the documents in the collection.
  - Update: allows us to update the existing document.
  - Delete: drop the existing document.
- Http methods are used in order to make request. The different methods used are:
  - GET: Gets all the documents in the collection.
  - POST: Creating new documents (products) in the collection.
  - PATCH: Updating the existing documents in the collection.
  - DELETE: Deleting the documents.
- Routes are defined for the respective requests and URLS are created.
  `/products`
  `/products/12345`
- These routes are then tested in any API testing application .in this case, POSTMAN.
- This is repeated for all the resources needed for the application like User model, Reviews model, Orders model, views model etc.

3.2 MVC Architecture

Model Views Controller is an architecture that separates business logic from presentation logic.it helps in refactoring of the code into a clean code and makes navigation and debugging easy.



Source: MDN docs.

The flow of request-response cycle in MVC architecture is as follows:

- The request is sent to the router of the API.
- From there, it goes to the Controller that is the application logic of the architecture.
- Now the business and presentational logics are separated. The controller goes to Models that is the business logic and gets the relevant data.
- Then it goes to Views that is the presentation logic to get the required data.
- Finally, the response is sent to the client.
- Now, we refactor our created API into MVC architecture by following these steps:
- Create a Controller folder and refactor all the application code into .js files here for example product Controller.js,userController.js etc.it is done by importing the resource model and the http requests.

productController.js

```
const Product = require('./../models/productModel');

exports.getAllproducts = async (req, res) => {
  try {
const products=await Product.find();

    // SEND RESPONSE
    res.status(200).json({
      status: 'success',
      results: products.length,
      data: {
        products
      }
    });
  } catch (err) {
    res.status(404).json({
      status: 'fail',
      message: err
    });
  }
};
```

- .now, create a routes folder and import all the data from controllers.

```
const express = require('express');
const productController = require('./../controllers/productController');

const router = express.Router();

router
  .route('/')
  .get(productController.getAllproducts)
  .post(productController.createProduct);

router
  .route('/:id')
  .get(productController.getProduct)
  .patch(productController.updateProduct)
  .delete(productController.deleteProduct);

module.exports = router;
```

- The views folder is created and the entire front-end rendering is done here.

4. FRONT END RENDERING

4.1 Client side rendering

This is the process of consuming the created API from the client-side.the website is created at client side using technologies like: React, Angular.js etc.

4.2 Server side rendering

This is the process of building the website on the server itself using templates like Pug, EJS, Handlebars etc. We can use any one of the template and consume our API using Axios to make API calls. This is what we have used in our example.
/overview.pug
/products.pug
/reviews.pug

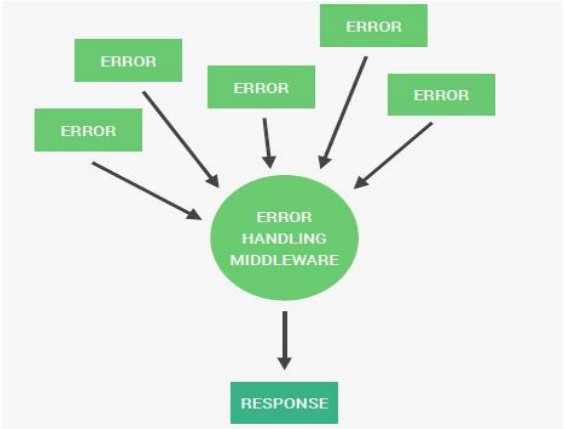
- The advantage of using axios is that we can easily use JavaScript logic without having to learn a new language.
- The AXIOS API call can be as below:

signup.js

```
import axios from "axios";  15.4K (gzipped: 5.2K)
import {
    showAlert
} from "./alert";
Complexity is 5 Everything is cool!
export const signup = async (data) => {
    try {
        const res = await axios({
            method: "POST",
            url: "http://localhost:8000/api/v1/users/signup",
            data
        });
        console.log(data);
        //res.data here is the json data we sent to API
        if (res.data.status === "success") {
            showAlert("success", "You are a member now!");
            window.setTimeout(() => {
                location.assign("/");
            }, 1500);
        }
    } catch (err) {
        showAlert('error', err.response.data.message);
    }
};
```

5. ERROR HANDLING

Express provides an extensive error handling mechanism. We can use this to our advantage by making a global middleware that handles all our error.



Source: Jonas schmedtmann course files.

All the errors related to databse, tokens, unhandled routes can be handled here.one can make a separate errorController.js file and import it wherever handling of errors is necessary. This error handler is called in the next () function which then reroutes it to our global middleware where express handles them.

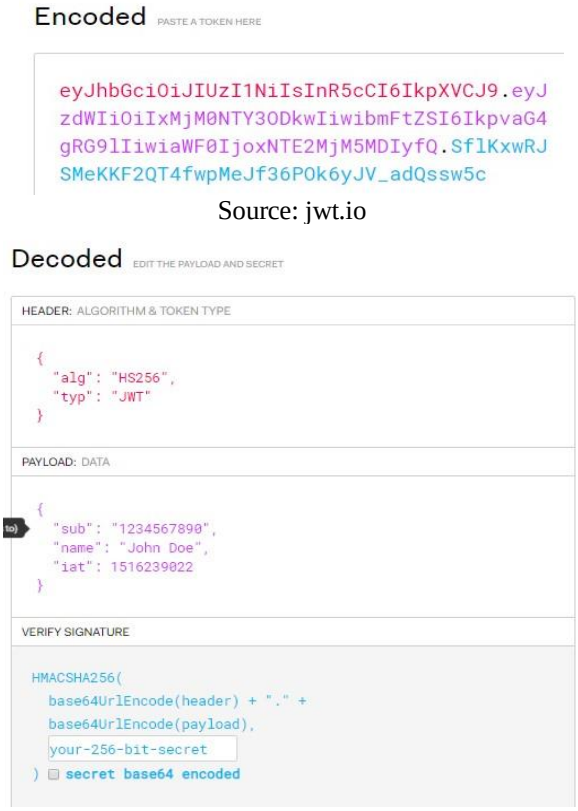
```
    fn(req, res, next).catch(next);
  };
}


app.get('*', wrapAsync(async function(req, res) {
  await new Promise(resolve => setTimeout(() => resolve(), 50
));
  throw new Error('Oppss!! Something went wrong');
}));


app.use(function(error, req, res, next) {
  res.json({ message: error.message });
```

Source: https://appdividend.com/2018/09/16/express-error-handling-example-tutorial/

6. AUTHENTICATION WITH JWT

JSON Web Tokens can be used for authentication, authorization in our API.first; let's see what JWT is.JWT is a unique token that is made up of:

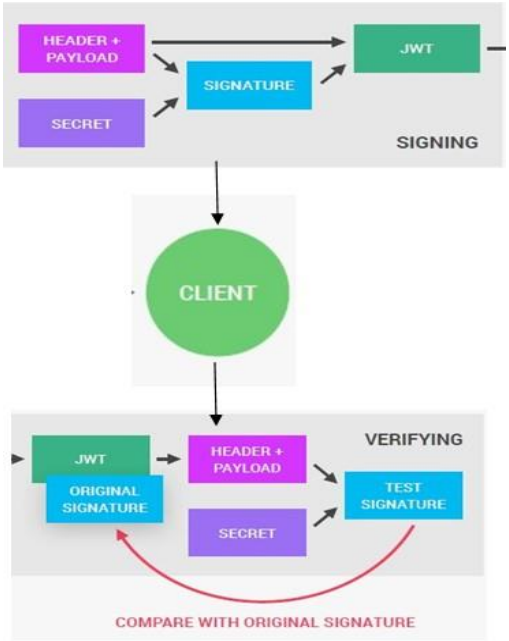
- Header: the authorization header
- Payload: data on which we are running the signature.
- Secret key: our own secret key which we created
- Signature: unique key generated from above.



Source: jwt.io

6.1 The working of JWT

Header and payload together with our secret key is combined and a signature is created. using this JWT is created and assigned to the user. When the user tries to Login using the JWT, a test signature is created and compared with the user's key. If it matches, then the user is logged in. If and only if the test signature matches the user's signature, only then is it authenticated.



Source: Jonas schmedtmann course files

6.2 Using JWT for Authentication

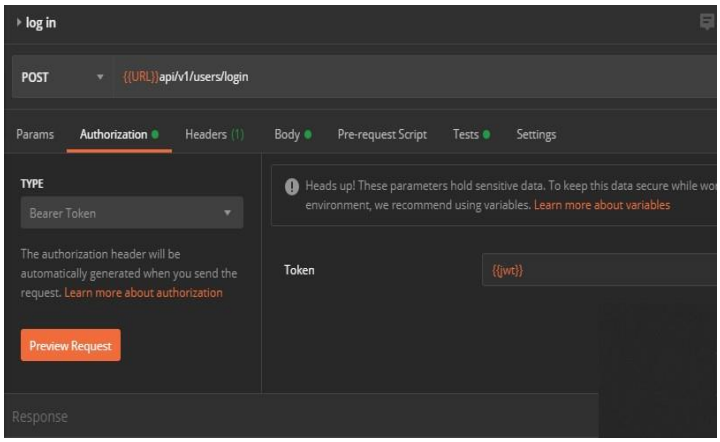
The steps involved in logging in our users with JWT are:

- JWT is required from NPM package.
- Creating a function for sending and storing the token. The tokens are stored in either local storage or cookies.
- Storing the tokens in cookies does not pose a threat as we set the cookie to be passed only on secure connections.

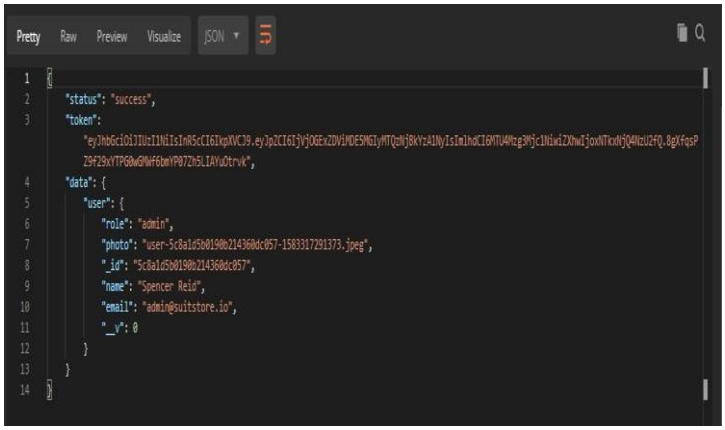
```
const createSendToken = (user, statusCode, res) => {
    const token = signToken(user._id);
    const cookieOptions = {
        expires: new Date(
            Date.now() + process.env.JWT_COOKIE_EXPIRES_IN * 24 * 60 * 60 * 1000
        ),
        httpOnly: true
    };
    if (process.env.NODE_ENV === "production") cookieOptions.secure = true;

    res.cookie("jwt", token, cookieOptions);
```

Source: Jonas schmedtmann course files.

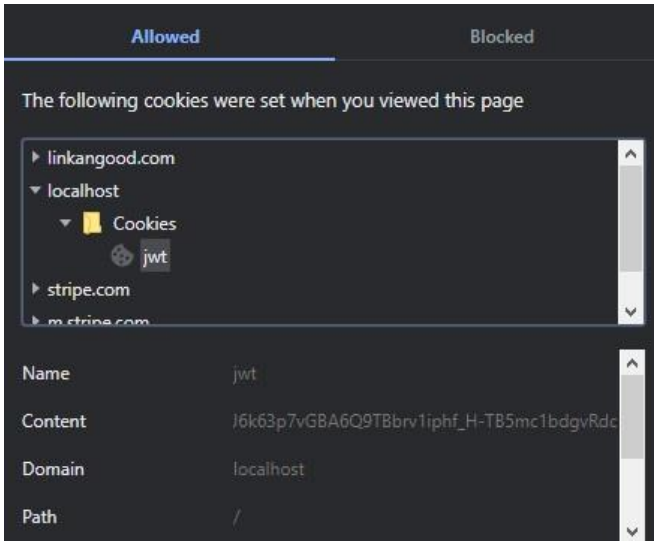
- The user passwords are stored in mongoDB by encrypting the using an NPM package called Crypto. This uses SHA algorithm.
- The user's model is also designed so as to not display password while performing user related CRUD operations.
- Now the token is sent in POSTMAN by using "Bearer Token" header. We are sending the tokens with headers because GET requests don't have a body.



- Once we log in, the following response is produced with our unique JWT and the user's information without the password.



- The JWT is stored in the cookie of the website.



• While logging out an empty JWT is sent because our cookie cannot be modified or deleted [7].

6.3 Authorization with JWT

If the JWT is valid, it gives access to some protected routes. We can authorize some of the operations such as updating and deleting of products only to Administrators.
The users can be authorized to view their orders, account etc.

7. CONCLUSION

Thus, integrating REST services with JWT authentication will provide a dependable and scalable backend to any e-commerce business. We have implemented the disused model with some additional API features and security by creating an online store. The findings were engaging and the development process as stated was easy and precise.

8. REFERENCES

[1] Vatrapu, Sidhartha Reddy, "Design and Implementation of E-Commerce Site for Online Shopping" (2014). All Capstone Projects. 79. https://opus.govst.edu/capstones/79
[2] Gefen, David. (2000). Gefen, D.: E-commerce: the role of familiarity and trust. OMEGA 28(6), 725-737. Omega. 28. 725-737.
[3] Statista Research Department,Amazon: number of active customer accounts worldwide 1997-2015, https://www.statista.com/statistics/237810/number-of-active-amazon-customer-accounts-worldwide/
[4] Mithilesh Tarkar,Ameya Parker, "APIs and Restful APIs" Published in International Journal of Trend in Scientific Research and Development (ijtsrd), ISSN: 2456-6470, Volume-2 | Issue-5 , August 2018, http://www.ijtsrd.com/computer-science/database/15797/apis-and-restful-apis/mithilesh-tarka
[5] https://restfulapi.net/
[6] wikipedia
[7] Jonas Schmedtmann course, https://github.com/jonasschmedtmann
[8] Nodejs docs, https://nodejs.org/uk/docs/guides/event-loop-timers-and-nexttick/.
[9] https://www.tutorialspoint.com/nodejs/nodejs_event_loop.htm.
[10] https://saegeullee.github.io/nodejs/what-is-express
[11] https://www.dineshonjava.com/what-is-rest-and-rest-architecture-and-rest-constraints/
[12] https://www.freecodecamp.org/news/introduction-to-mongoose-for-mongodb-d2a7aa593c57/

