

## AWT authentication

### AWT authentication

#### 1. Checking Get Request :

GET http://localhost:8092/ Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings

TYPE  
No Auth

Here there will be no authentication

This request does not use any authorization. [Learn more about authorization](#)

Body Cookies Headers (11) Test Results Status: 403 Forbidden Time: 1272 ms Size: 455 B Save

Pretty Raw Preview Visualize JSON

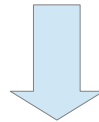
```
1 {  
2   "timestamp": "2020-08-15T12:09:09.444+00:00",  
3   "status": 403,  
4   "error": "Forbidden",  
5   "message": "",  
6   "path": "/"  
7 }
```

Request is Forbidden

## AWT authentication

## 2. Generating JWT token

```
/**
 * generateToken method is used to generate the token
 *
 * @param authRequest
 * @return String
 * @throws Exception
 */
@PostMapping("/authenticate")
public String generateToken(@RequestBody AuthRequest authRequest) throws Exception {
    try {
        authenticationManager.authenticate(
            new UsernamePasswordAuthenticationToken(authRequest.getUserName(), authRequest.getPassword())
        ) catch (Exception ex) {
            throw new Exception("invalid username/password");
        }
        return jwtUtil.generateToken(authRequest.getUserName());
    }
}
```

[illegible]

### 3.Accessing API with JWT token

The screenshot shows the Postman interface for an API request. The request method is GET and the URL is http://localhost:8092/. The Authorization tab is selected, and the type is set to Bearer Token. The token value is pasted into the Token field. A yellow callout bubble points to the Bearer Token dropdown with the text "Selecting the Bearer Token". Another yellow callout bubble points to the token value with the text "Generated Token is Pasted Here". Below the token field, there is a note: "The authorization header will be automatically generated when you send the request. [Learn more about authorization](#)". The Body tab is selected, and the response is displayed in the Pretty view. A yellow callout bubble points to the response body with the text "Authentication Success". The response body contains the text "1 Welcome to Blog !!". A blue arrow points from the response body to a code block on the right.

GET http://localhost:8092/ Send Save

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies Code

TYPE Bearer Token Token

eyJhbGciOiJIUzI1NiJ9.eyJzdWwiOiI1c2VyMyIsImV4cCI6MTU5Nzg5NzU4MCwiaWf...

Selecting the Bearer Token

Generated Token is Pasted Here

The authorization header will be automatically generated when you send the request. [Learn more about authorization](#)

Body Cookies Headers

Authentication Success

1 Welcome to Blog !!

```

    /**
     * creating a get mapping that retrieves the initial string content
     */
    @GetMapping("/")
    public String welcome() {
        return "Welcome to Blog !!";
    }

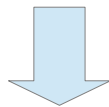
```

## Crud Operation

### Performing CRUD operations:

### Inserting Data into DataBase :

```
/**
 * creating post mapping that post the book detail in the database
 *
 * @param books
 * @return int
 */
@PostMapping("/books")
private int saveBook(@RequestBody Books books) {
    booksService.saveOrUpdate(books);
    return books.getBookid();
}
```



The screenshot shows an API client interface with the following details:

- Method: POST
- URL: http://localhost:8092/books
- Body: A JSON object with the following fields:

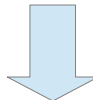
```
{
  "bookid": "0982",
  "bookname": "Programming with Java",
  "author": "E. Balagurusamy",
  "price": "350",
  "content": "Java is a powerful general-purpose programming language. It is used to develop desktop and mobile applications, big data processing, embedded systems, and so on."
}
```
- Response: 200 OK

A yellow callout bubble points to the JSON body with the text: "Inserting Data into the DB Using POST method".

## Crud Operation

### Getting All Data From Database:

```
/**
 * creating a get mapping that retrieves all the books detail from the database
 *
 * @return List<String>
 */
@GetMapping("/book")
private List<String> getAllBooks() {
    List<Books> booksList = booksService.getAllBooks();
    List<String> bookDetails = booksList.stream().map(book ->{
        StringBuffer ff = new StringBuffer();
        String bookDetail = book.getBookname()+"^^^^^^"+book.getAuthor()+"\n";
        return bookDetail;
    }).collect(Collectors.toList());
    return bookDetails;
}
```



The screenshot shows a REST client interface with a GET request to `http://localhost:8092/book/`. The response is a JSON array of book details. A yellow callout bubble points to the response data, stating: "Fetching Data from the DB Using GET method".

Request: GET `http://localhost:8092/book/` Send

Params Authorization Headers (9) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2   "bookid": 982,
3   "bookname": "Programming with Java",
4   "author": "E. Balagurusamy",
5   "price": 350,
6   "content": "Java is a powerful general-purpose programming language. It is used to develop desktop and mobile applications, bi
7   processing, embedded systems, and so on."
8 }
```

Body Cookies Headers (11) Test Results Status: 200 OK Time: 42 ms Size: 507 B Save

Pretty Raw Preview Visualize JSON

```
1 [
2   "Programming with Java^^^^^^E. Balagurusamy\n",
3   "Core and Advance Java^^^^^^R. Nageswara Rao\n",
4   "Data Structures and Algorithms in Java^^^^^^Robert Lafore\n"
5 ]
```

Fetching Data from the DB Using GET method

### Fetching Particular ID from DataBase :

```

/**
 * creating a get mapping that retrieves the detail of a specific book
 *
 * @param bookid
 * @return List<String>
 */
@GetMapping("/book/{bookid}")
private List<String> getBooks(@PathVariable("bookid") int bookid) {
    List<String> stringList = new ArrayList<>();
    Books book = booksService.getBooksById(bookid);
    StringBuilder ff = new StringBuilder();
    String bookDetail = "\t"+book.getBookname()+"\t"+"^^^^^^"+" \t"+book.getAuthor();
    stringList.add("*****Author Name :"+book.getAuthor()+"*****\r\n");
    stringList.add("*****Book Name :"+book.getBookname()+"*****\r\n");
    stringList.add("*****Book Price :"+book.getPrice()+"*****\r\n");
    stringList.add(bookDetail+"\n");
    stringList.add(Const.TEXT_CONTENT);
    stringList.add(Const.TEXT_CONTENT+'\n');
    return stringList;
}

```



## Crud Operation

GET http://localhost:8092/book/982

Send Save

Params Authorization Headers (7) Body Pre-request Script Test Results Cookies Cc

none form-data x-www-form-urlencoded raw binary

**Fetching Particular ID from the DB Using GET method**

Status: 200 OK Time: 37 ms Size: 4.92 KB Save Response

Pretty Raw Preview Visualize JSON

```
1
2 *****Author Name :E. Balagurusamy*****\r\n",
3 *****Book Name :Programming with Java*****\r\n",
4 *****Book Price :350*****\r\n",
5 "\tProgramming with Java\t*****\tE. Balagurusamy\n",
6 "Java requires that each variable be initialized. \nSome older languages such as C, allow variables to go uninitialized, which can cause random failures with mysterious bugs.\nJava requires that each method declares a return type-the method should always return a value, except if its return type is void. This also prevents bugs\nJava comes with a large set of classes and methods, the Java API that can be used without having to develop as much code \nUnlike C, Java primitive types, such as int, are always the same size in the number of bits which helps achieve cross-platform compatibility\nJava used to be thought of as being slower than C, but that's become less important in recent years because computers are faster\nJava has exception-handling that requires a programmer to handle error-conditions such as Input/Output errors\nCode compiled on one Java platform can be run on other platforms that support Java without modification of either the source-code nor the byte-code. This means that a person can make a Java program for a Windows computer and have it run a Linux computer or a Mac computer\nJava requires that each variable be initialized. \nSome older languages such as C, allow variables to go uninitialized, which can cause random failures with mysterious bugs.\nJava requires that each method declares a return type-the method should always return a value, except if its return type is void. This also prevents bugs\nJava comes
```

THANK YOU !!!