

</ Inventory  
Management  
System Backend  
Dev. using Flask

/>

CPSC  
449

Amanda Shohdy, Brandy Nguyen,  
Huy Nguyen, Michael Baldo,  
Carlos Hernandez

# User Registration

```
# User registration
@app.route('/register', methods=['POST'])
def register():
    if not request.json or 'username' not in request.json or 'password' not in request.json or 'email' not in request.json:
        return jsonify({'error': 'Invalid input'}), 400

    username = request.json['username']
    password = request.json['password']
    email = request.json['email']

    if username in users:
        return jsonify({'error': 'Username already exists'}), 400

    if not is_valid_email(email):
        return jsonify({'error': 'Invalid email format'}), 400

    if len(password) < 8 or not re.search(r"[A-Z]", password) or not re.search(r"[a-z]", password) or not re.search(r"[0-9]", password):
        return jsonify({'error': 'Password must be at least 8 characters long and contain at least one uppercase letter, one lowercase letter, and one digit'}), 400

    users[username] = {
        'password': password,
        'email': email
    }

    return jsonify({'message': 'User registered successfully'}), 201
```

POST http://127.0.0.1:5000/register

Params Authorization Headers (9) Body Scripts

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw

```
1 {
2   "username" : "testUser",
3   "password" : "testPassword1",
4   "email" : "test@gmail.com"
5 }
```

Body Cookies Headers (5) Test Results

{ } JSON Preview Visualize

```
1 {
2   "message": "User registered successfully"
3 }
```

Check all fields are filled out

Extract username, password, and email

Check username is unique

Check email and password formats

Register user

1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1

# User Login

```
# User login
@app.route('/login', methods=['POST'])
def login():
    if not request.json or 'username' not in request.json or 'password' not in request.json:
        return jsonify({'error': 'Invalid input'}), 400

    username = request.json['username']
    password = request.json['password']

    if username not in users or users[username]['password'] != password:
        return jsonify({'error': 'Invalid username or password'}), 401

    token = jwt.encode(
        { 'username' : username, 'exp' : datetime.datetime.utcnow() + datetime.timedelta(hours=1) },
        app.config['SECRET_KEY'],
        algorithm="HS256"
    )
    print("Encoded token type:", type(token))

    session['username'] = username
    response = jsonify({'message': 'Login successful', 'token': token})
    response.set_cookie('username', username, httponly=True, max_age=1800)

    return response, 200
```

POST http://127.0.0.1:5000/login

Params Authorization Headers (10) Body Scripts

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw

```
1 {
2   "username" : "testUser",
3   "password" : "testPassword1"
4 }
```

Body Cookies (2) Headers (8) Test Results

{ } JSON Preview Visualize

```
1 {
2   "message": "Login successful",
3   "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpzZW50b3R5IiwiaWF0IjoxNjU0MjU0MjU0"
4 }
```

Check all fields are filled out

Extract username and password

Validate credentials

Issue JWT token

Establish session and cookie

1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1

# User Logout

```
# User logout
@app.route('/logout', methods=['POST'])
def logout():
    session.pop('username', None)
    response = jsonify({'message': 'Logout successful'})
    response.set_cookie('username', '', expires=0) # Clear the cookie
    return response, 200
```

End session

Output success message

Clear cookies

The screenshot shows a REST client interface. The top section displays a POST request to `http://127.0.0.1:5000/logout`. The 'Body' tab is selected, showing a JSON payload: `{ "username": "testUser", "password": "testPassword1" }`. The bottom section shows the response body, also in JSON: `{ "message": "Logout successful" }`. The interface includes tabs for Params, Authorization, Headers, Body, Cookies, Headers, and Test Results.

# Create Inventory Items

```

@app.route('/inventory', methods=['POST'])
@token_required
def create_inventory_item(currentUser):
    required_fields = ['name', 'description', 'quantity', 'price']
    if not request.json or not all(field in request.json for field in required_fields):
        return jsonify({'error': 'Invalid input'}), 400

    if 'name' in request.json and not isinstance(request.json['name'], str):
        return jsonify({'error': 'Name must be a string'}), 400

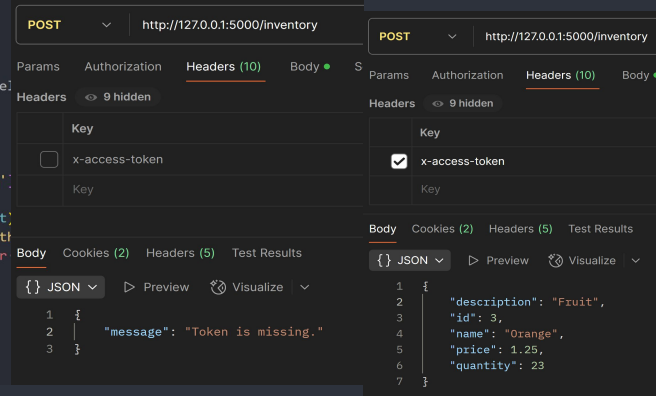
    if 'description' in request.json and not isinstance(request.json['description'], str):
        return jsonify({'error': 'Description must be a string'}), 400

    if 'quantity' in request.json and not isinstance(request.json['quantity'], int):
        return jsonify({'error': 'Quantity must be an integer and cannot be less than 0'}), 400

    if 'price' in request.json and not isinstance(request.json['price'], float) or request.json['price'] < 0:
        return jsonify({'error': 'Invalid price format'}), 400

    inventory_id = max(item['id'] for item in inventory) + 1 if inventory else 1
    inventory_item = {**request.json, 'id': inventory_id}
    inventory.append(inventory_item)
    return jsonify(inventory_item), 201

```



JWT token required

Check all fields are filled out

## Validate name, description, quantity, and price fields

```
Generate item id number and inventory item, then add item to inventory
```

1 0 1 1    0 1 1    0 1    1 0 1 1 0 0 1    1 0    1 1 0 1 1    0 1 1    0 1    1 1 0 1 1 0    1 1 0 1 1 1    1 1 0 1

# Update Inventory Item

```
@app.route('/inventory/<int:item_id>', methods=['PUT'])
@token_required
def update_inventory_item(currentUser, item_id):
    item = find_item(item_id)
    if item is None:
        return jsonify({'error': 'Item not found'}), 404
    if not request.json:
        return jsonify({'error': 'Invalid input, request body must be JSON'}), 400
    if 'name' in request.json and not isinstance(request.json['name'], str):
        return jsonify({'error': 'Name must be a string'}), 400
    if 'description' in request.json and not isinstance(request.json['description'], str):
        return jsonify({'error': 'Description must be a string'}), 400
    if 'quantity' in request.json and not isinstance(request.json['quantity'], int) or request.json['quantity'] < 0:
        return jsonify({'error': 'Quantity must be an integer and cannot be less than 0'}), 400
    if 'price' in request.json and not isinstance(request.json['price'], float) or request.json['price'] < 0:
        return jsonify({'error': 'Invalid price format'}), 400
    item.update(request.json)
    return jsonify(item)
```

JWT token required

Find item or error if item is not found

Validate fields name, description, quantity, and price

Update item

PUT

▼

http://127.0.0.1:5000/inventory/2

ParamsAuthorizationHeaders (10)Body ●

☐ none

☐ form-data

☐ x-www-form-urlencoded

1 {

2     "name": "Grape",

3     "description": "Fruit",

4     "quantity": 28,

5     "price": 2.50

6 }

BodyCookies (2)Headers (5)Test Results

{ } JSON ▼

▶ Preview

🔗 Visualize ▼

1 {

2     "description": "Fruit",

3     "id": 2,

4     "name": "Grape",

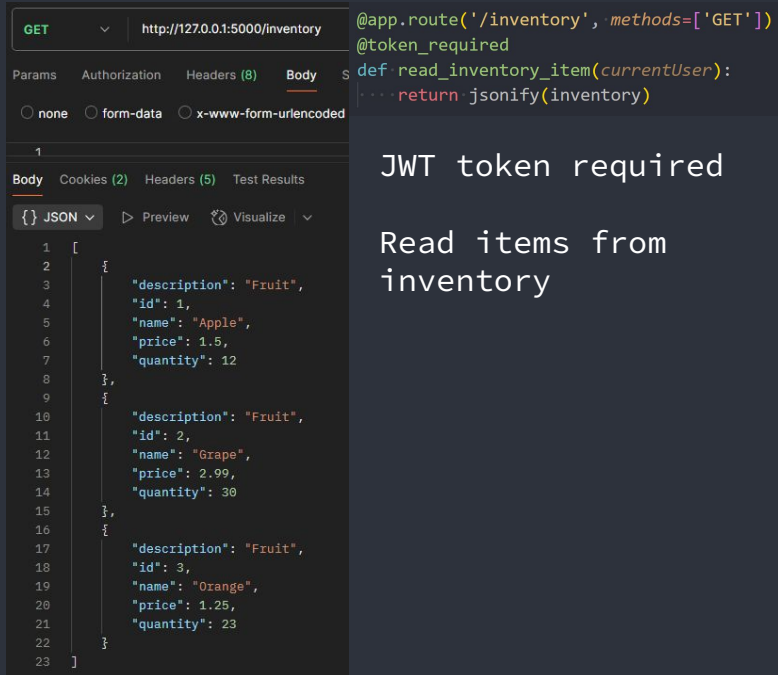
5     "price": 2.5,

6     "quantity": 28

7 }

1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 0 1

# READ & DELETE



```
GET http://127.0.0.1:5000/inventory
```

Params Authorization Headers (8) **Body**

☐ none ☐ form-data ☐ x-www-form-urlencoded

1

**Body** Cookies (2) Headers (5) Test Results

{ } JSON Preview Visualize

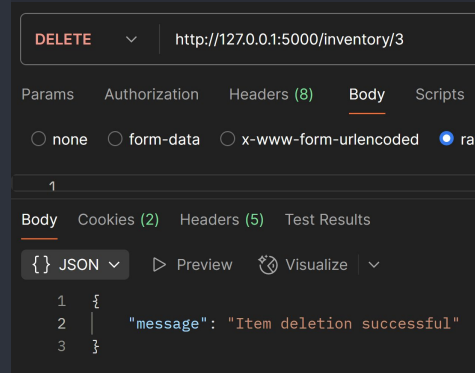
```
1 [
2   {
3     "description": "Fruit",
4     "id": 1,
5     "name": "Apple",
6     "price": 1.5,
7     "quantity": 12
8   },
9   {
10    "description": "Fruit",
11    "id": 2,
12    "name": "Grape",
13    "price": 2.99,
14    "quantity": 30
15  },
16  {
17    "description": "Fruit",
18    "id": 3,
19    "name": "Orange",
20    "price": 1.25,
21    "quantity": 23
22  }
23 ]
```

@app.route('/inventory', methods=['GET'])  
@token\_required  
def read\_inventory\_item(currentUser):  
 return jsonify(inventory)

JWT token required

Read items from  
inventory

```
@app.route('/inventory/<int:item_id>', methods=['DELETE'])  
@token_required  
def delete_inventory_item(currentUser, item_id):  
    item = find_item(item_id)  
    if item is None:  
        return jsonify({'error': 'Item not found'}), 404  
    inventory.remove(item)  
    return jsonify({'message': 'Item deletion successful'}), 200
```



```
DELETE http://127.0.0.1:5000/inventory/3
```

Params Authorization Headers (8) **Body** Scripts

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw

1

**Body** Cookies (2) Headers (5) Test Results

{ } JSON Preview Visualize

```
1 {
2   "message": "Item deletion successful"
3 }
```

JWT token required

Find item or error  
if item not found

Remove item from  
inventory

# Technology Used

Python



Postman



Imports:

- From flask
  - Flask
  - jsonify
  - session
  - request
- re
- jwt
- wraps
- datetime

1 0 1 1 0 1 1 0 1 1 0 1 1 0 0 1 1 0 1 1 0 1 1 0 1 1 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1



# THANKS!

Questions?

1 0 1 1   0 1 1   0 1   1 0 1 1 0 0 1   1 0   1 1 0 1 1   0 1 1   0 1   1 1 0 1 1 0   1 1 0 1 1 1   1 1 0 1