</ Grocery Store Inventory Management />

CPSC 449

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

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

Project Purpose

- Acts as the backend API for inventory management
- Built using FastPI + MongoDB
- Handles user authentication, role based access, and CRUD operations on 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

Core Technologies

- FastAPI for API routes
- MongoDB for data persistence
- Pydantic for data validation
- JWT for session authentication
- Dotenv for environment variable handling.

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

User Authentication System

- Register and Login routes
- Password validation rules:
  - > 8 characters, uppercase, lowercase, digits, and special characters
- JWT token issued at login, stored as HTTP only cookie
- Logout route deletes the cookie

```python
def validate_password(password: str):
    if len(password) < 8:
        return "Password must be at least 8 characters long."
    if not re.search(r'\d', password):
        return "Password must contain at least one digit."
    if not re.search(r'[A-Z]', password):
        return "Password must contain at least one uppercase letter."
    if not re.search(r'[a-z]', password):
        return "Password must contain at least one lowercase letter."
    if not re.search(r'[\W_]', password):
        return "Password must contain at least one special character."
    return None
```

```python
@app.post("/register", response_model=UserOut)
def register_user(user: UserCreate):
    if db.users.find_one({"username": user.username}):
        raise HTTPException(status_code=400, detail="Username already taken.")
    if db.users.find_one({"email": user.email}):
        raise HTTPException(status_code=400, detail="Email already taken.")
    password_error = validate_password(user.password)
    if password_error:
        raise HTTPException(status_code=400, detail=password_error)

    new_user = user.dict()
    db.users.insert_one(new_user)
    created_user = db.users.find_one({"username": user.username})
    return {"id": str(created_user["_id"]), "username": created_user["username"]}
```

```python
@app.post("/login")
def login_user(user: UserLogin, response: Response):
    db_user = db.users.find_one({"username": user.username})
    if not db_user or db_user["password"] != user.password:
        raise HTTPException(status_code=401, detail="Invalid username or password.")

    token = create_access_token(data={"sub": user.username})
    response.set_cookie(key="access_token", value=token, httponly=True, secure=False, samesite='lax')
    return {"message": "Login Successful."}

@app.post("/logout")
def logout(response: Response):
    response.delete_cookie("access_token")
    return {"message": "Logged out successfully"}
```

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

Access Control

- get_current_user() validates JWT and pulls user from DB

```python
def get_current_user(request: Request):
    token = request.cookies.get("access_token")
    if not token:
        raise HTTPException(status_code=401, detail="Not authenticated")
    try:
        payload = jwt.decode(token, SECRET_KEY, algorithms=[ALGORITHM])
        username = payload.get("sub")
        if not username:
            raise HTTPException(status_code=401, detail="Invalid token")
    except JWTError:
        raise HTTPException(status_code=401, detail="Invalid Token or Expired Token")

    user = db.users.find_one({"username": username})
    if not user:
        raise HTTPException(status_code=404, detail="User not found")
    return user
```

```python
def admin_required(current_user: dict = Depends(get_current_user)):
    if current_user.get("role") != "admin":
        raise HTTPException(status_code=403, detail="Admin access required")
    return current_user
```

- admin_required() checks if current user has “admin” role
- Inventory create, update, and delete routes are admin only

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

Admin Only

- Only users with admin role can access this route:
  - POST /inventory
  - PUT /inventory/{item_id}
  - DELETE /inventory/{item_id}

- If the user is not an admin, an error is raised automatically

```python
@app.delete("/inventory/{item_id}")
def delete_item(item_id: int, current_user: dict = Depends(admin_required)):
    result = db.inventory.delete_one({"id": item_id})
    if result.deleted_count == 0:
        raise HTTPException(status_code=404, detail="Item not found.")
    return {"message": "Item deleted successfully"}
```

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

Data Models

- UserCreate: email, username, password, role
- UserLogin: username, password
- ItemCreate: name, description, price, quantity
- ItemOut: id, name, description, price, quantity

```python
class UserLogin(BaseModel):
    username: str
    password: str

class UserCreate(BaseModel):
    email : EmailStr
    username: str
    password: str
    role: str = "user"

class UserOut(BaseModel):
    id: str
    username: str

class ItemCreate(BaseModel):
    name: str
    description: str
    price: float
    quantity: int

class ItemOut(BaseModel):
    id: str
    name: str
    description: str
    price: float
    quantity: int
```

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

Inventory Endpoints

Endpoint	Method	Description	Access
/inventory	POST	Add Items	Admin
/inventory	GET	Get all items	Authenticated
/inventory/{item_id}	GET	Get single item	Authenticated
/inventory/{item_id}	PUT	Update Item	Admin
/inventory/{item_id}	DELETE	Delete Item	Admin

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

Add Inventory

```python
@app.post("/inventory", response_model=ItemOut)
def add_item(item: ItemCreate, current_user: dict = Depends(admin_required)):
    item_dict = item.dict()
    item_dict["user_id"] = str(current_user["_id"])
    item_dict["id"] = get_next_item_id()

    if db.inventory.find_one({"name": item_dict["name"], "user_id": item_dict["user_id"]}):
        raise HTTPException(status_code=400, detail="Item already exists for this user.")

    db.inventory.insert_one(item_dict)
    return {
        "id": item_dict["id"],
        "name": item_dict["name"],
        "description": item_dict["description"],
        "price": item_dict["price"],
        "quantity": item_dict["quantity"],
        "user_id": item_dict["user_id"]
    }
```

- Validates user admin
- Checks if item exists
- Inserts item in MongoDB
- Returns inserted item

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

Technology Used

Python

Postman

MySQL

MongoDB

FastAPI

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

THANKS!

Questions?

1011 011 01 1011001 10 11011 011 01 110110 110111 1101