

Deliverable 4
IMPLEMENTASI AWAL
II3160 - Integrated Systems Technology



Disusun oleh:
Theresia Ivana Marella Siswahyudi - 18223126

Program Studi Sistem Dan Teknologi Informasi
Sekolah Teknik Elektro Dan Informatika
Institut Teknologi Bandung
2025

DAFTAR ISI

DAFTAR ISI.....	2
LATAR BELAKANG.....	3
DESAIN ARSITEKTUR IMPLEMENTASI.....	4
IMPLEMENTASI DOMAIN MODEL.....	5
A. Aggregate Root: InventoryItem.....	5
B. Value Objects.....	5
C. Entities.....	5
IMPLEMENTASI APPLICATION LAYER.....	7
A. Repository: InventoryRepositoryInMemory.....	7
B. Service: InventoryService.....	7
IMPLEMENTASI API LAYER.....	9
A. Struktur Umum API.....	9
B. Endpoint Admin Gudang (/admin/...).....	9
C. Endpoint Open Host Service (/ohs/...).....	10
D. Endpoint Manager (/manager/...).....	11
E. Integrasi dengan Pydantic Schema.....	11
MENJALANKAN APLIKASI DAN LINGKUNGAN EKSEKUSI.....	12
1. Instalasi dan Persiapan Lingkungan.....	12
2. Menjalankan Aplikasi.....	12
3. Dokumentasi & Pengujian API.....	13
PENGUJIAN API DENGAN SWAGGER UI, POSTMAN & PYTEST.....	14
A. Pengujian Menggunakan Swagger UI.....	15
B. Pengujian Menggunakan Postman.....	32
C. Pengujian Menggunakan pytest.....	40
KESIMPULAN.....	42

LATAR BELAKANG

Dalam pengembangan Sistem Manajemen Pergudangan (Warehouse Management System/WMS), tahap implementasi merupakan fase penting untuk mewujudkan rancangan domain dan arsitektur yang sebelumnya telah dianalisis. Pengelolaan stok sebagai bagian inti dari proses pergudangan memegang peranan krusial dalam menjamin kelancaran rantai pasok. Aktivitas seperti pencatatan barang masuk dan keluar, pengecekan ketersediaan barang, serta pemantauan kondisi stok harus dikelola secara akurat agar tidak menimbulkan kesalahan operasional yang berdampak pada efisiensi distribusi dan kualitas layanan.

Sebagai langkah lanjutan dari perancangan domain, *bounded context Inventory Control* diimplementasikan dengan mengacu pada prinsip *Domain-Driven Design* (DDD). Aggregate `InventoryItem`, beserta Value Object seperti `SKU`, `Quantity`, dan `Threshold`, serta Entity seperti `Reservation` dan `StockMove`, diterjemahkan ke dalam kode Python untuk membangun fondasi logika bisnis yang konsisten dan terstruktur. Lapisan aplikasi juga dikembangkan melalui `InventoryService` dan `InventoryRepositoryInMemory` untuk menyediakan berbagai use case pengelolaan stok, sementara API berbasis FastAPI disediakan untuk memungkinkan integrasi dengan modul lain melalui Open Host Service (OHS).

Tahap implementasi ini juga disertai proses pengujian untuk memastikan fungsionalitas berjalan sesuai kebutuhan operasional. Pengujian dilakukan secara manual menggunakan Swagger UI dan Postman untuk memverifikasi setiap endpoint, serta secara otomatis menggunakan pytest untuk memastikan aturan bisnis seperti penambahan stok, pengurangan stok, reservasi, dan deteksi low stock bekerja dengan benar. Dengan adanya implementasi dan validasi menyeluruh ini, sistem memperoleh fondasi teknis yang kuat untuk mendukung pengembangan fitur lanjutan pada tahap selanjutnya.

DESAIN ARSITEKTUR IMPLEMENTASI

Arsitektur implementasi modul Inventory Control dibangun dengan pendekatan berlapis yang mengacu pada prinsip Domain-Driven Design (DDD). Tujuannya adalah untuk memisahkan logika domain inti dari detail teknis implementasi, sehingga sistem menjadi lebih terstruktur, mudah dipahami, dan mudah dikembangkan pada tahap berikutnya. Pada implementasi ini digunakan tiga lapisan utama, yaitu Domain Layer, Application Layer, dan API Layer.

Lapisan Domain Layer berisi model inti bisnis yang merepresentasikan aturan pengelolaan stok. Struktur utama pada lapisan ini adalah aggregate `InventoryItem` yang menjadi pusat perilaku domain, termasuk operasi penambahan stok, pengurangan stok, penyesuaian stok, dan manajemen reservasi. Komponen pendukung seperti Value Object (`SKU`, `Quantity`, `Threshold`, `Batch`) dan Entity (`Reservation`, `StockMove`) juga diimplementasikan pada lapisan ini untuk memastikan bahwa setiap aturan bisnis diatur secara konsisten dan tidak bergantung pada detail teknis seperti database maupun API.

Lapisan Application Layer mengimplementasikan use case pengelolaan stok melalui kelas `InventoryService`. Lapisan ini bertanggung jawab untuk mengoordinasikan pemanggilan logika domain dan mengelola penyimpanan data melalui `InventoryRepositoryInMemory`. Repository ini menggunakan struktur penyimpanan berbasis in-memory yang memudahkan proses pengembangan awal tanpa ketergantungan pada sistem basis data. Application Layer bertindak sebagai penghubung antara domain dan antarmuka API, memastikan bahwa setiap operasi seperti reserve, release, increase, dan decrease dijalankan sesuai urutan dan aturan bisnis yang benar.

Lapisan paling luar adalah API Layer, yang diimplementasikan menggunakan framework FastAPI melalui file `main.py`. Lapisan ini mendefinisikan endpoint yang dapat diakses oleh berbagai aktor seperti admin gudang, sistem Order, sistem Inbound, dan Manajer. API disusun berdasarkan kelompok fungsi (admin, OHS, dan manager) serta menggunakan Pydantic untuk validasi input dan penyusunan Data Transfer Object (DTO). Pendekatan ini memastikan bahwa data yang masuk ke dalam sistem tervalidasi dengan baik, serta setiap respons yang diberikan API konsisten dan mudah diproses oleh modul lain. Dengan struktur arsitektur berlapis tersebut, modul Inventory Control memiliki fondasi implementasi yang bersih, modular, dan siap dikembangkan pada tahap integrasi berikutnya.

Tabel 1. Pemetaan Layer

Layer	Folder	Fungsi
Domain Layer	<code>src/domain/</code>	Aturan bisnis inti, aggregate, entity, value object
Application Layer	<code>src/services/</code>	Use case, repository, koordinasi domain
API Layer	<code>src/main.py</code>	Endpoint FastAPI, input/output HTTP
DTO / Schema Layer	<code>src/schemas/</code>	Validasi request/response menggunakan Pydantic

IMPLEMENTASI DOMAIN MODEL

Implementasi domain model bertujuan untuk merepresentasikan aturan bisnis inti dari pengelolaan inventori sesuai prinsip Domain-Driven Design (DDD). Domain model ditempatkan di dalam folder `src/domain/` dan terdiri dari komponen-komponen utama seperti Aggregate Root, Value Object, serta Entity, yang bekerja bersama untuk memastikan konsistensi data dan perilaku sistem.

A. Aggregate Root: `InventoryItem`

Aggregate utama pada modul ini adalah `InventoryItem`, yang berfungsi sebagai pusat logika bisnis terkait pengelolaan stok barang. Aggregate ini menyimpan informasi mengenai jumlah stok tersedia (`on_hand`), stok yang sedang dipesan (`reserved`), ambang batas stok minimum (`threshold`), serta catatan pergerakan stok (`moves`). Beberapa operasi penting yang diimplementasikan pada aggregate ini meliputi:

- `increase()` – menambah jumlah stok barang berdasarkan `Quantity`.
- `decrease()` – mengurangi stok yang tersedia, dengan validasi agar tidak melebihi stok `available`.
- `reserve()` – menyimpan stok untuk pesanan tertentu dan menambah daftar `reservation`.
- `release()` – membatalkan `reservation` tertentu dan mengembalikannya ke stok `available`.
- `adjust()` – menyesuaikan stok secara manual (positif atau negatif).

Setiap perubahan menerapkan *invariant checks* untuk memastikan bahwa nilai `on_hand`, `reserved`, dan `available` tetap valid dan konsisten.

B. Value Objects

Value Objects digunakan untuk merepresentasikan nilai-nilai yang tidak memiliki identitas unik dan ditentukan oleh nilainya.

- SKU - Digunakan untuk merepresentasikan kode produk. Dibuat immutable dan divalidasi agar tidak kosong.
- Quantity - Mewakili jumlah stok beserta satuannya (`uom`). Mendukung operasi penjumlahan (`add`) dan pengurangan (`sub`) dengan validasi UOM dan mencegah hasil negatif.
- Threshold - Digunakan untuk menentukan batas minimal stok sebelum dianggap low stock.
- Batch - Menyimpan informasi batch produksi dan tanggal kedaluwarsa jika diperlukan di masa depan.

Penggunaan value object ini memastikan konsistensi data dan mencegah duplikasi logika validasi.

C. Entities

Entitas digunakan untuk objek yang memiliki identitas unik selama daur hidupnya. Identitas ini tidak berubah meskipun atribut lain pada entitas tersebut mengalami perubahan seiring berjalannya proses bisnis. Dengan kata lain, entitas diperlakukan sebagai objek yang tetap sama sepanjang waktu, bukan berdasarkan nilai-nilai atributnya, tetapi berdasarkan identity, biasanya berupa ID unik.

- Reservation - Digunakan untuk menyimpan stok yang dialokasikan untuk sebuah pesanan (order_id). Setiap reservation memiliki ID unik dan jumlah barang yang dicadangkan.
- StockMove - Mewakili pergerakan stok, baik itu masuk (IN), keluar (OUT), maupun penyesuaian (ADJUST). Informasi seperti jumlah, tipe pergerakan, alasan, dan timestamp dicatat sebagai histori.

Kedua entitas ini membantu menjaga histori dan memastikan integritas operasi reserve, release, increase, dan decrease.

Dengan diterapkannya domain model yang kuat, sistem dapat menjamin bahwa setiap operasi yang terjadi pada inventori mengikuti aturan bisnis yang benar dan terhindar dari inkonsistensi data. Model ini juga menjadi pondasi bagi Application Layer dan API Layer sehingga seluruh alur kerja Inventory Control dapat berjalan secara konsisten dan terstruktur.

IMPLEMENTASI APPLICATION LAYER

Application Layer berfungsi sebagai penghubung antara domain model dan antarmuka API. Lapisan ini mengatur alur eksekusi use case, mengoordinasikan pemanggilan metode pada aggregate, serta menangani penyimpanan data melalui repository. Dalam implementasi modul Inventory Control, Application Layer ditempatkan pada folder `src/services/` dan terdiri dari dua komponen utama, yaitu repository dan service.

A. Repository: `InventoryRepositoryInMemory`

Repository berfungsi sebagai abstraksi mekanisme penyimpanan data sehingga domain maupun API tidak bergantung pada detail teknis seperti database atau file. Pada tahap implementasi awal ini, digunakan repository berbasis in-memory, yang menyimpan data menggunakan struktur Python dictionary. Walaupun sederhana, pendekatan ini efektif untuk memastikan fungsi domain dapat dieksekusi dan diuji tanpa ketergantungan sistem eksternal. Repository menyediakan beberapa operasi dasar:

Tabel 2. Method pada `InventoryRepositoryInMemory`

Method	Fungsi
<code>list_all()</code>	Mengembalikan semua item inventori yang tersimpan
<code>get_by_id()</code>	Mengambil item berdasarkan ID
<code>get_by_sku()</code>	Mengambil item berdasarkan SKU
<code>save()</code>	Menyimpan atau memperbarui item dalam repository

Dengan memisahkan penyimpanan lewat repository, implementasi ini dapat dengan mudah dikembangkan ke database sesungguhnya pada milestone berikutnya tanpa mengubah logika domain.

B. Service: `InventoryService`

`InventoryService` merupakan komponen inti pada Application Layer yang bertanggung jawab menjalankan use case pengelolaan stok. Service ini memanfaatkan repository untuk mengambil atau menyimpan data, kemudian memanggil metode pada aggregate `InventoryItem` untuk menerapkan aturan bisnis yang sesuai. Service menyediakan berbagai operasi penting, di antaranya:

Tabel 3. Use Case Application Layer

Kategori Use Case	Nama Use Case	Deskripsi Fungsi
Admin Gudang	<code>create_item</code>	Membuat item baru dengan stok awal, satuan (uom), dan batas minimum stok.

	<code>list_items</code>	Menampilkan seluruh item inventori yang tersedia di sistem.
	<code>get_item</code>	Mengambil informasi lengkap item berdasarkan SKU.
	<code>set_threshold</code>	Mengubah nilai minimum stok (threshold) untuk suatu item.
Inbound / Outbound	<code>increase_stock</code>	Menambah stok ketika barang masuk (goods received / inbound).
	<code>decrease_stock</code>	Mengurangi stok ketika barang keluar (outbound / shipment).
	<code>adjust_stock</code>	Melakukan koreksi stok manual, baik penambahan maupun pengurangan (stock opname).
Order / Reservation	<code>reserve_stock</code>	Melakukan reservasi stok untuk suatu order sehingga stok disisihkan.
	<code>release_reservation</code>	Membatalkan reservasi dan mengembalikan stok ke available.
Monitoring	<code>get_availability</code>	Memberikan informasi ketersediaan stok: on_hand, reserved, dan available.
	<code>get_low_stock_items</code>	Menampilkan daftar item yang berada di bawah batas minimum stok (low stock).

Setiap use case ini mengatur interaksi antara API layer dan domain model, memastikan bahwa aturan bisnis dalam domain dijalankan secara konsisten dan semua perubahan inventori tersimpan dengan benar melalui repository.

Dengan struktur Application Layer yang terpisah ini, kode menjadi lebih modular, mudah diuji, dan mendukung prinsip separation of concerns. Lapisan ini juga memastikan bahwa domain tetap bersih dari detail teknis, dan API dapat berinteraksi dengan domain melalui antarmuka yang jelas dan terstruktur.

IMPLEMENTASI API LAYER

API Layer berfungsi sebagai antarmuka antara sistem Inventory Control dengan dunia luar, baik pengguna manusia (misalnya admin gudang melalui tool seperti Postman) maupun sistem lain (seperti modul Order, Inbound, atau Outbound). Pada implementasi ini, API dibangun menggunakan framework FastAPI dan didefinisikan di dalam file `src/main.py`. Lapisan ini memetakan setiap endpoint HTTP ke use case yang telah disediakan oleh `InventoryService`.

A. Struktur Umum API

Aplikasi FastAPI diinisialisasi dengan membuat instance FastAPI dan kemudian mendefinisikan beberapa kelompok endpoint berdasarkan aktor atau konteks pengguna:

- Endpoint Admin → prefix `/admin/...`
Digunakan oleh admin atau staf gudang untuk mengelola master data item dan pengaturan stok.
- Endpoint OHS (Open Host Service) → prefix `/ohs/...`
Digunakan oleh sistem lain (Order, Inbound, Outbound) untuk berinteraksi dengan Inventory Control.
- Endpoint Manager → prefix `/manager/...`
Digunakan oleh manajer untuk memantau kondisi stok dan low stock.

Setiap endpoint menggunakan model Pydantic dari `src/schemas/inventory.py` sebagai request body dan response DTO, sehingga data yang masuk dan keluar API tervalidasi secara otomatis.

B. Endpoint Admin Gudang (`/admin/...`)

Kelompok endpoint ini memanfaatkan use case kategori Admin Gudang dari `InventoryService`.

Tabel 4. Endpoint Admin Gudang

Endpoint	Method	Use Case	Deskripsi
<code>/admin/items</code>	POST	<code>create_item</code>	Membuat item baru dengan stok awal dan threshold.
<code>/admin/items</code>	GET	<code>list_items</code>	Mengambil seluruh item inventori.
<code>/admin/items/{sku}</code>	GET	<code>get_item</code>	Mengambil detail satu item berdasarkan SKU.
<code>/admin/items/{sku}/threshold</code>	POST	<code>set_threshold</code>	Mengubah batas minimum stok (threshold) suatu item.
<code>/admin/items/{sku}/adjust</code>	POST	<code>adjust_stock</code>	Melakukan penyesuaian stok manual (stock opname).

Di dalam implementasi, endpoint menerima request dalam bentuk `CreateItemRequest`, `SetThresholdRequest`, atau `AdjustStockRequest`, lalu memanggil method service yang sesuai dan mengembalikan response dalam bentuk `InventoryItemDto`.

C. Endpoint Open Host Service (`/ohs/...`)

Kelompok endpoint ini merepresentasikan API yang seolah-olah dipakai oleh bounded context lain untuk berinteraksi dengan Inventory Control. Fungsinya lebih ke operasional harian (inbound, outbound, reservasi).

Tabel 5. *Endpoint OHS*

Endpoint	Method	Use Case	Deskripsi
<code>/ohs/{sku}/increase</code>	POST	<code>increase_stock</code>	Menambah stok saat barang masuk (inbound).
<code>/ohs/{sku}/decrease</code>	POST	<code>decrease_stock</code>	Mengurangi stok saat barang keluar (outbound).
<code>/ohs/{sku}/reserve</code>	POST	<code>reserve_stock</code>	Melakukan reservasi stok untuk order tertentu.
<code>/ohs/{sku}/release</code>	POST	<code>release_reservation</code>	Membatalkan reservasi tertentu berdasarkan reservation ID.
<code>/ohs/availability/{sku}</code>	GET	<code>get_availability</code>	Mengambil informasi <code>on_hand</code> , <code>reserved</code> , dan <code>available</code> untuk sebuah SKU.
<code>/ohs/{sku}/reservations*</code>	GET	(akses ke <code>item.reservations</code>)	Menampilkan daftar reservasi untuk suatu SKU. (*endpoint tambahan yang kamu implementasikan).

Body request untuk endpoint ini menggunakan schema seperti `IncreaseStockRequest`, `DecreaseStockRequest`, `ReserveStockRequest`, dan `ReleaseReservationRequest`. API kemudian mengembalikan data baik dalam bentuk `InventoryItemDto` atau `InventoryStats`, tergantung kebutuhan.

D. Endpoint Manager (/manager/...)

Kelompok endpoint ini menyediakan informasi untuk kebutuhan monitoring level manajerial.

Tabel 6. Endpoint Manager

Endpoint	Method	Use Case	Deskripsi
/manager/low-stock	GET	get_low_stock_items	Mengembalikan daftar item yang berada di bawah threshold (low stock).

Endpoint ini mengembalikan list `InventoryItemDto` sehingga manajer dapat melihat SKU, stok on hand, reserved, available, serta informasi threshold dan status `low_stock`.

E. Integrasi dengan Pydantic Schema

Setiap endpoint menggunakan Pydantic model dari `schemas/inventory.py` untuk memisahkan representasi domain dengan bentuk data yang ditransfer melalui API. Contohnya:

- `CreateItemRequest` untuk input pembuatan item.
- `IncreaseStockRequest`, `DecreaseStockRequest`, `AdjustStockRequest` untuk operasi stok.
- `InventoryItemDto` dan `InventoryStats` untuk response yang dikembalikan ke klien.

Pemakaian DTO ini memastikan bahwa domain model (`InventoryItem`, `Reservation`, dll.) tidak terekspos langsung ke dunia luar, sekaligus menjaga konsistensi format data untuk dokumentasi otomatis di Swagger UI.

MENJALANKAN APLIKASI DAN LINGKUNGAN EKSEKUSI

Sebelum aplikasi dijalankan, dilakukan proses instalasi dan persiapan lingkungan kerja untuk memastikan seluruh dependency dapat berfungsi dengan baik. Proyek ini menggunakan Python dan menjalankan aplikasi melalui FastAPI dengan server ASGI Uvicorn. Adapun langkah-langkah instalasi dan eksekusi adalah sebagai berikut.

1. Instalasi dan Persiapan Lingkungan

- 1) Clone repository

```
git clone https://github.com/meerancor33/II3160_TB_18223126_WMS.git  
cd II3160_TB_18223126_WMS
```

- 2) Buat virtual environment

```
python -m venv .venv
```

- 3) Aktifkan virtual environment

```
## PowerShell  
.venv\Scripts\Activate  
  
## Git Bash / MinGW  
source .venv/Scripts/activate  
  
## macOS / Linux  
source .venv/bin/activate  
  
## Jika berhasil, terminal akan menampilkan prefix:  
(.venv)
```

- 4) Install dependencies

```
pip install -r requirements.txt
```

2. Menjalankan Aplikasi

Setelah seluruh dependency terpasang, aplikasi dijalankan menggunakan perintah:

```
uvicorn src.main:app --reload
```

Jika berjalan dengan sukses, server akan menampilkan pesan:

```
Uvicorn running on http://127.0.0.1:8000  
Application startup complete.
```

Pada implementasi ini, ketika pengguna membuka URL <http://127.0.0.1:8000>, sistem secara otomatis melakukan redirect ke halaman dokumentasi API (Swagger UI) untuk memudahkan proses pengujian endpoint.

3. Dokumentasi & Pengujian API

FastAPI menyediakan dokumentasi otomatis yang dapat digunakan untuk menguji seluruh endpoint tanpa perlu menggunakan alat eksternal.

a) Swagger UI (Direkomendasikan)

Digunakan untuk melihat, mencoba, dan menguji seluruh endpoint API (GET, POST, DELETE, dsb).

<http://127.0.0.1:8000/docs>

<http://localhost:8000/docs>

b) ReDoc UI (Alternatif)

Menampilkan dokumentasi dalam format read-only dengan tampilan yang lebih terstruktur.

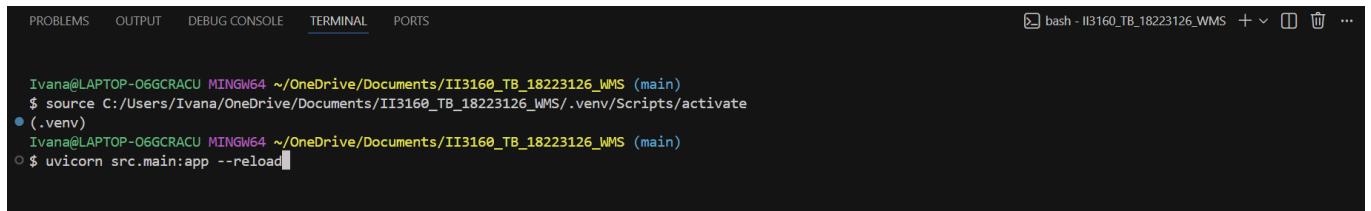
<http://127.0.0.1:8000/redoc>

<http://localhost:8000/redoc>

Melalui kedua dokumentasi ini, pengguna dapat memverifikasi seluruh fungsi API yang telah diimplementasikan, termasuk operasi admin gudang, inbound/outbound, reservasi, dan monitoring low stock.

PENGUJIAN API DENGAN SWAGGER UI, POSTMAN & PYTEST

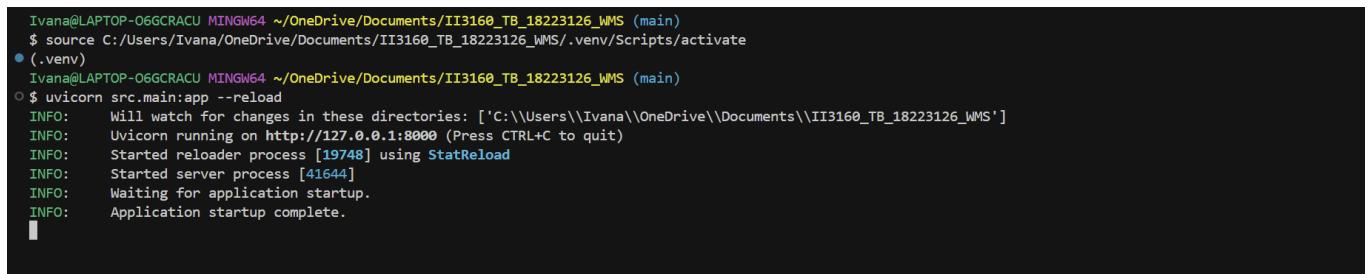
Pengujian dilakukan untuk memastikan bahwa seluruh endpoint pada modul Inventory Control berfungsi sesuai dengan aturan bisnis yang telah ditetapkan pada domain model. Proses pengujian dilakukan menggunakan dua pendekatan, yaitu pengujian manual melalui Swagger UI dan Postman. Kedua alat ini memungkinkan pengembang atau pengujii untuk memvalidasi request dan response secara langsung, memastikan bahwa alur kerja sistem berjalan sebagaimana mestinya.



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
bash - II3160_TB_18223126_WMS + × ⌂ ...
```

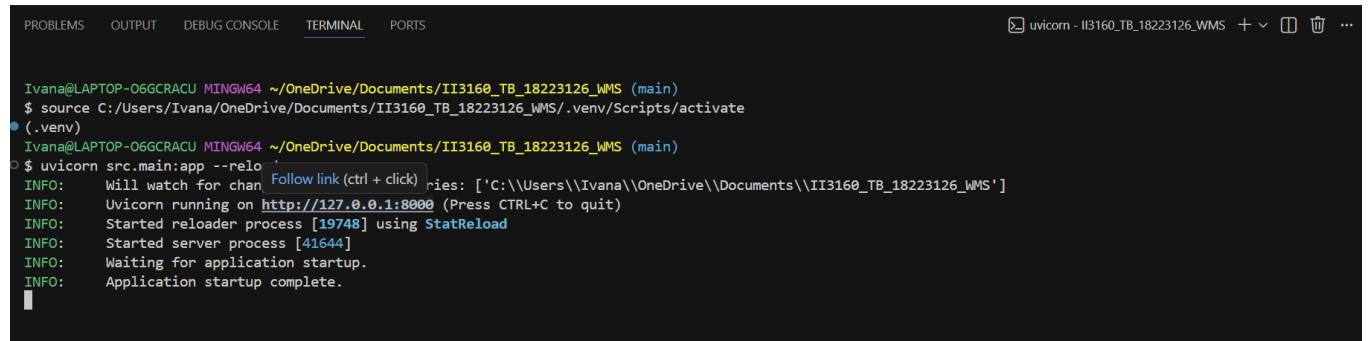
```
Ivana@LAPTOP-06GCRACU MINGW64 ~/OneDrive/Documents/II3160_TB_18223126_WMS (main)
$ source C:/Users/Ivana/OneDrive/Documents/II3160_TB_18223126_WMS/.venv/Scripts/activate
● (.venv)
Ivana@LAPTOP-06GCRACU MINGW64 ~/OneDrive/Documents/II3160_TB_18223126_WMS (main)
○ $ uvicorn src.main:app --reload
```

Gambar 1. Langkah Pertama untuk Menjalankan Aplikasi



```
Ivana@LAPTOP-06GCRACU MINGW64 ~/OneDrive/Documents/II3160_TB_18223126_WMS (main)
$ source C:/Users/Ivana/OneDrive/Documents/II3160_TB_18223126_WMS/.venv/Scripts/activate
● (.venv)
Ivana@LAPTOP-06GCRACU MINGW64 ~/OneDrive/Documents/II3160_TB_18223126_WMS (main)
○ $ uvicorn src.main:app --reload
INFO: Will watch for changes in these directories: ['C:\\Users\\Ivana\\OneDrive\\Documents\\II3160_TB_18223126_WMS']
INFO: Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO: Started reloader process [19748] using StatReload
INFO: Started server process [41644]
INFO: Waiting for application startup.
INFO: Application startup complete.
```

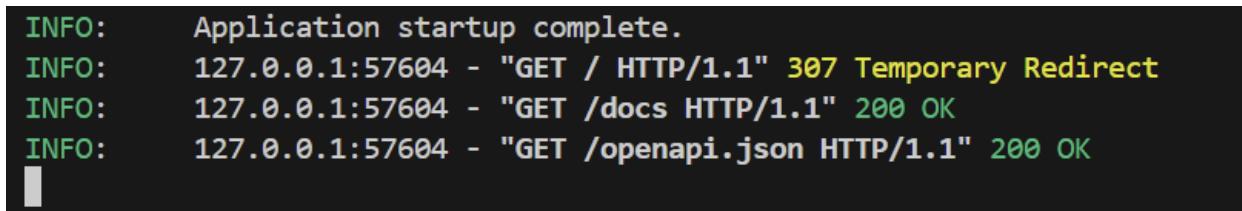
Gambar 2. Tampilan ketika Aplikasi Berhasil Berjalan



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
uvicorn - II3160_TB_18223126_WMS + × ⌂ ...
```

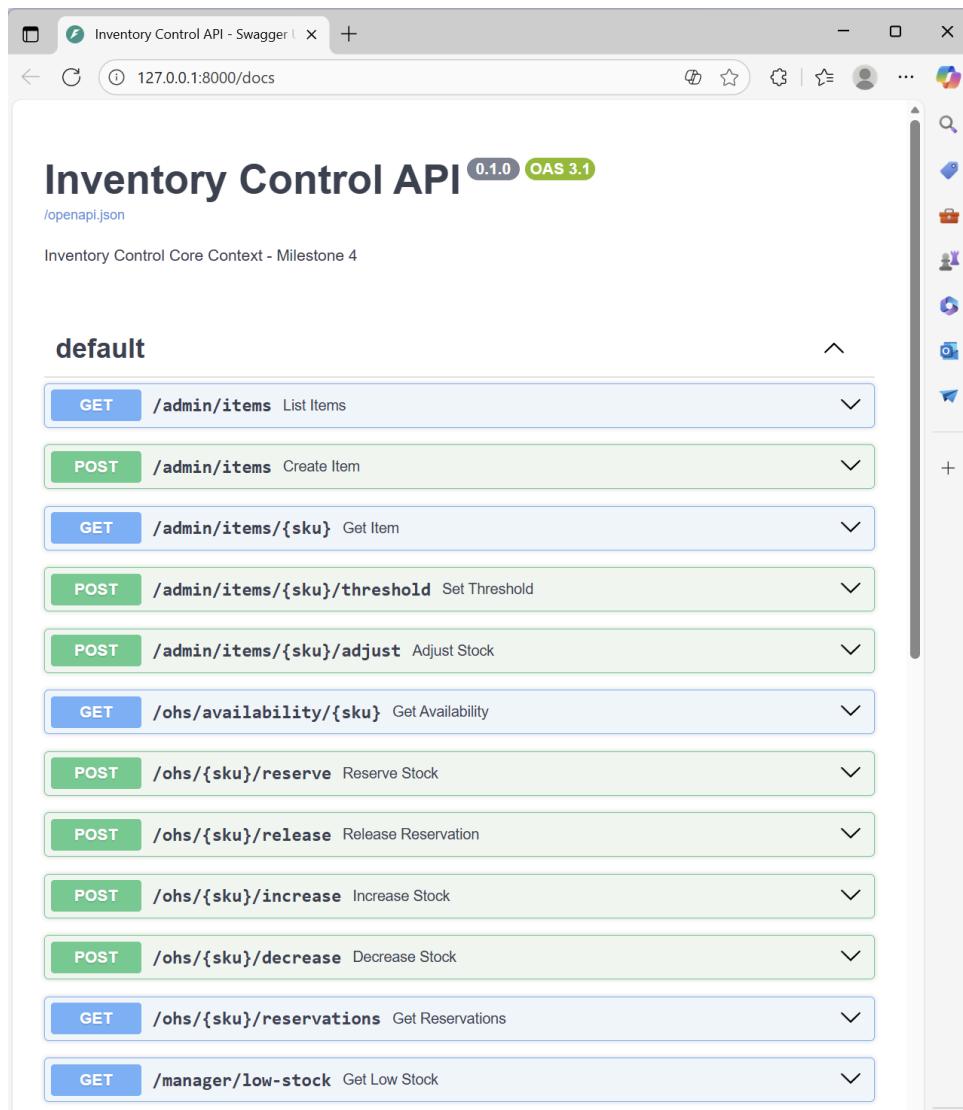
```
Ivana@LAPTOP-06GCRACU MINGW64 ~/OneDrive/Documents/II3160_TB_18223126_WMS (main)
$ source C:/Users/Ivana/OneDrive/Documents/II3160_TB_18223126_WMS/.venv/Scripts/activate
● (.venv)
Ivana@LAPTOP-06GCRACU MINGW64 ~/OneDrive/Documents/II3160_TB_18223126_WMS (main)
○ $ uvicorn src.main:app --relo ...
INFO: Will watch for chan Follow link (ctrl + click) ries: ['C:\\Users\\Ivana\\OneDrive\\Documents\\II3160_TB_18223126_WMS']
INFO: Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO: Started reloader process [19748] using StatReload
INFO: Started server process [41644]
INFO: Waiting for application startup.
INFO: Application startup complete.
```

Gambar 3. Tampilan Tautan untuk Redirect ke Swagger UI



```
INFO: Application startup complete.
INFO: 127.0.0.1:57604 - "GET / HTTP/1.1" 307 Temporary Redirect
INFO: 127.0.0.1:57604 - "GET /docs HTTP/1.1" 200 OK
INFO: 127.0.0.1:57604 - "GET /openapi.json HTTP/1.1" 200 OK
```

Gambar 4. Tampilan Ketika Fitur Redirect Berhasil Berjalan dan Swagger UI Berhasil Dimuat



Gambar 5. Tampilan Swagger UI Inventory Control

A. Pengujian Menggunakan Swagger UI

Swagger UI merupakan fitur bawaan FastAPI yang menyediakan antarmuka interaktif untuk melihat dan mencoba seluruh endpoint. Melalui halaman ini, pengguna dapat mengirim request secara langsung, mengisi request body, dan mengamati response yang dihasilkan.

Membuat item baru

POST /admin/items

Tampilan awal:

The screenshot shows the initial view of the API documentation for the `POST /admin/items` endpoint. It includes sections for Parameters, Request body (with a required field), and Responses (including 200 and 422 status codes).

Request body (required)

```
{  
  "sku": "SKU-001",  
  "initial_qty": 0,  
  "min_qty": 0,  
  "max_qty": 0  
}
```

Responses

Code	Description	Links
200	Successful Response	No links
422	Validation Error	No links

Example Value | Schema

```
{  
  "id": "string",  
  "sku": "string",  
  "on_hand": 0,  
  "initial_qty": 0,  
  "available": 0,  
  "min_qty": 0,  
  "max_qty": 0,  
  "low_stock": true,  
  "reservations": []  
}
```

Example Value | Schema

```
{  
  "detail": [  
    {  
      "loc": [  
        "string",  
        0  
      ],  
      "msg": "string",  
      "type": "string"  
    }  
  ]  
}
```

Tampilan Try It Out:

The screenshot shows the `Try It Out` interface for the `POST /admin/items` endpoint. It includes fields for Parameters, Request body (with a required field), and an Execute button.

Request body (required)

```
{  
  "sku": "SKU-001",  
  "initial_qty": 0,  
  "min_qty": 0,  
  "max_qty": 0  
}
```

Execute

Tampilan Edit Request Body:

Request body required

Edit Value | Schema

```
{  
  "sku": "SKU-001",  
  "initial_qty": 100,  
  "uom": "pcs",  
  "min_qty": 10  
}
```

Tampilan Setelah Execute:

Responses

Curl

```
curl -X 'POST' \  
  'http://127.0.0.1:8000/admin/items' \  
  -H 'accept: application/json' \  
  -H 'Content-Type: application/json' \  
  -d '{  
    "sku": "SKU-001",  
    "initial_qty": 100,  
    "uom": "pcs",  
    "min_qty": 10  
  }'
```

Request URL

```
http://127.0.0.1:8000/admin/items
```

Tampilan jika Mencoba Menambah Item dengan ID SKU yang Sama:

Server response

Code Details

400 Error: Bad Request
Undocumented

Response body

```
{"detail": "Item with this SKU already exists"}  
)
```

Download

Response headers

```
content-length: 46  
content-type: application/json  
date: Mon, 17 Nov 2025 13:01:15 GMT  
server: Unicorn
```

Melihat daftar item

GET /admin/items

Tampilan awal:

The screenshot shows the API documentation for the `GET /admin/items` endpoint. It includes sections for Parameters (none), Responses (Code: 200, Description: Successful Response, Media type: application/json), and Example Value (a JSON array of item objects). The example value is partially obscured by a black rectangle.

Tampilan Try It Out:

The screenshot shows the "Try It Out" interface for the `GET /admin/items` endpoint. It features a "Parameters" section (none) and a large blue "Execute" button.

Tampilan Edit Request Body: Tidak ada body

Tampilan Setelah Execute:

The screenshot shows the results of the executed `GET /admin/items` request. It includes sections for Responses (Curl command, Request URL, Server response), Code (200, Details), and Response body (JSON array of items). The response body is identical to the example value shown in the initial view. A "Download" button is also present.

Melihat detail item berdasarkan SKU

GET /admin/items/{sku}

Tampilan awal:

The screenshot shows the 'Tampilan awal:' (Initial View) of the API documentation for the 'Get Item' endpoint. It includes sections for 'Parameters' (with a required 'sku' parameter), 'Responses' (200: Successful Response, media type application/json), and a large redacted 'Example Value' section.

Tampilan Try It Out:

The screenshot shows the 'Tampilan Try It Out:' (Try It Out View) of the API documentation. It features a 'Parameters' section with a 'sku' field containing 'SKU-001', an 'Execute' button, and a 'Cancel' button.

Tampilan Edit Request Body:

The screenshot shows the 'Tampilan Edit Request Body:' (Edit Request Body View) of the API documentation. It displays a 'Parameters' section with a 'sku' field containing 'SKU-001', an 'Execute' button, and a 'Cancel' button.

Tampilan Setelah Execute:

The screenshot shows the 'Tampilan Setelah Execute:' (Results After Execution) of the API documentation. It includes sections for 'Parameters' (sku: SKU-001), 'Responses', 'Curl' command, 'Request URL' (http://127.0.0.1:8000/admin/items/SKU-001), 'Server response', and a detailed 'Response body' section. The response body is as follows:

```
{
  "id": "7ad313bc-28f7-4862-8ef7-d93bde05cfac",
  "sku": "SKU-001",
  "on_hand": 300,
  "reserved": 0,
  "available": 300,
  "min_qty": 10,
  "max_qty": 100,
  "low_stock": false,
  "reservations": []
}
```

Below the response body, there are 'Response headers' with values: content-length: 165, content-type: application/json, date: Mon, 17 Nov 2025 13:04:44 GMT, and server: unicorn.

Melakukan peningkatan stok

POST /ohs/{sku}/increase

Tampilan awal:

The screenshot shows the API documentation for the `POST /ohs/{sku}/increase` endpoint. It includes sections for Parameters, Request body, and Responses.

Parameters: A parameter named `sku` is listed as required, with a type of string (path).

Request body: Required. Example value: `{"qty": 10, "reason": "INBOUND"}`. Media type: application/json.

Responses: 200 - Successful Response. Example value: `{"id": "string", "on_hand": 10, "reserved": 0, "available": 10, " uom": "string", "unit": "string", "low_stock": true, "reservations": []}`.

Tampilan Try It Out:

The screenshot shows the `Try It Out` interface for the `POST /ohs/{sku}/increase` endpoint. It has fields for Parameters and Request body, and a large Execute button.

Parameters: A parameter named `sku` is listed as required, with a type of string (path).

Request body: Required. Edit Value: `{"qty": 10, "reason": "INBOUND"}`. Media type: application/json.

Execute button.

Tampilan Edit Request Body:

Name	Description
sku * required string (path)	SKU-001

Request body required

Edit Value | Schema

```
{
  "qty": 10,
  "reason": "INBOUND"
}
```

Tampilan Setelah Execute:

Responses

Curl

```
curl -X 'POST' \
  'http://127.0.0.1:8000/obs/SKU-001/increase' \
  -H 'Content-Type: application/json' \
  -H 'Accept: application/json' \
  -d '{
    "qty": 10,
    "reason": "INBOUND"
}'
```

Request URL

http://127.0.0.1:8000/obs/SKU-001/increase

Server response

Code	Details
200	Response body <pre>{ "id": "760f10c9-28f7-4862-90ff-d93bed5cfac", "sku": "SKU-001", "on_hand": 110, "reason": "INBOUND", "available": 110, "min_qty": 10, "max_qty": 110, "low_stock": false, "reservations": [] }</pre> <p>Download</p> Response headers <pre>content-length: 165 content-type: application/json date: Mon, 17 Nov 2023 13:08:31 GMT server: unicorn</pre>

Melakukan pengurangan stok

POST /ohs/{sku}/decrease

Tampilan awal:

The screenshot shows the API documentation for the `POST /ohs/{sku}/decrease` endpoint. It includes sections for **Parameters**, **Request body**, and **Responses**.

Parameters: A table with one row for `sku` (required, string, path).

Request body: A schema example:

```
{
  "qty": 2,
  "reason": "CONSUME"
}
```

Responses: A table with one row for `200` (Successful Response, application/json). It shows the response schema:

```
{
  "id": "string",
  "on_hand": 1,
  "reserved": 0,
  "qty": 2,
  "use": "string",
  "qty": 2,
  "low_stock": true,
  "reservations": []
}
```

Tampilan Try It Out:

The screenshot shows the `Try It Out` interface for the same endpoint. It has fields for **Parameters** (sku: SKU-001) and **Request body** (with the same JSON schema as the documentation).

Tampilan Edit Request Body:

The screenshot shows the `Edit Request Body` interface. The `sku` parameter is set to `SKU-001`. The **Request body** field contains the following JSON:

```
{
  "qty": 20,
  "reason": "CONSUME"
}
```

Tampilan Setelah Execute:

Responses

```
Curl
curl -X POST \
http://127.0.0.1:8000/ohs/SKU-001/decrease \
-H 'accept: application/json' \
-H 'Content-Type: application/json' \
-d '{
    "qty": 20,
    "reason": "CONSUME"
}'
```

Request URL
http://127.0.0.1:8000/ohs/SKU-001/decrease

Server response

Code	Details
200	<p>Response body</p> <pre>{ "id": "74d31e0c-28f7-4862-90f7-d93b8e05cfac", "sku": "SKU-001", "on_hand": 0, "reserved": 0, "available": 0, "use": "pcx", "min_qty": 0, "lock": false, "reservations": [] }</pre> <p>Response headers</p> <pre>content-length: 163 content-type: application/json date: Mon, 17 Nov 2019, 13:11:03 GMT server: unicorn</pre>

[Download](#)

Melakukan reservasi stok

POST /ohs/{sku}/reserve

Tampilan awal:

POST /ohs/{sku}/reserve Reserve Stock

Parameters

Name	Description
sku * required	string (path)

Request body required

application/json

Example Value | Schema

```
{
    "order_id": "string",
    "qty": 0
}
```

Responses

Code	Description	Links
200	Successful Response	No links

Media type: application/json
Content Accept header

Example Value | Schema

```
{
    "id": "string",
    "sku": "string",
    "on_hand": 0,
    "reserved": 0,
    "available": 0
}
```

Tampilan Try It Out:

POST /ohs/{sku}/reserve Reserve Stock

Parameters

Name	Description
sku * required	string (path)

Request body required

application/json

Edit Value | Schema

```
{
    "order_id": "string",
    "qty": 0
}
```

Execute

Tampilan Edit Request Body:

Name	Description
sku * required string (path)	SKU-001

Request body required

Edit Value | Schema

```
{  
    "order_id": "string",  
    "qty": 10  
}
```

Tampilan Setelah Execute:

Responses

Curl

```
curl -X 'POST' \
  'http://127.0.0.1:8000/obs/SKU-001/reserve' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{  
    "order_id": "string",  
    "qty": 10  
}'
```

Request URL

http://127.0.0.1:8000/obs/SKU-001/reserve

Server response

Code	Details
200	Response body

```
{  
    "id": "7ad13e8c-28f7-4802-90f7-d93b8e05cfac",  
    "order_id": "string",  
    "on_hand": 0,  
    "available": 0,  
    "use": "pre",  
    "in_stock": true,  
    "low_stock": false,  
    "reservations": [  
        {"id": "170a0a32-938f-46fe-9b2b-113c2b34fc02",  
        "order_id": "string",  
        "qty": 10  
    ]  
}
```

Download

Response headers

```
content-length: 238  
content-type: application/json  
date: Mon, 20 Jun 2023 11:19:31 GMT  
server: unicorn
```

Melihat daftar reservasi stok

GET /ohs/{sku}/reservations

Tampilan awal:

The screenshot shows the API documentation for the '/ohs/{sku}/reservations' endpoint. It includes sections for 'Parameters', 'Responses', and two example responses (200 and 422). The 'Parameters' section shows a required parameter 'sku' of type string (path). The 'Responses' section shows a successful response (200) with a media type of application/json, and a validation error response (422) with a schema for validation errors.

Tampilan Try It Out:

The screenshot shows the 'Try It Out' interface for the '/ohs/{sku}/reservations' endpoint. It displays the same parameters and responses as the initial view, but includes a large blue 'Execute' button at the bottom.

Tampilan Edit Request Body:

The screenshot shows the 'Edit Request Body' interface for the '/ohs/{sku}/reservations' endpoint. It shows the 'sku' parameter set to 'SKU-001' and a large blue 'Execute' button below it.

Tampilan Setelah Execute: (saat belum ada reservasi)

The screenshot shows the results of the executed request. It includes a 'Curl' command, a 'Request URL' (http://127.0.0.1:8000/ohs/SKU-001/reservations), and a 'Server response'. The response is a 200 OK status with an empty JSON body and standard headers.

```
Curl: curl -X GET \
  "http://127.0.0.1:8000/ohs/SKU-001/reservations" \
  -H "Accept: application/json"
Request URL: http://127.0.0.1:8000/ohs/SKU-001/reservations
Server response
Code: 200
Details: Response body
[]

Response headers
content-length: 2
content-type: application/json
date: Mon, 17 Nov 2025 13:16:16 GMT
server: unicorn
```

(saat sudah ada reservasi):

```

curl -X 'GET' \
  'http://127.0.0.1:8000/ohs/SKU-001/reservations' \
  -H 'Content-Type: application/json'

```

Request URL: <http://127.0.0.1:8000/ohs/SKU-001/reservations>

Server response

Code	Details
200	Response body

```

[{"id": "170ba125-0387-46fe-9b2b-113c034fc02", "sku_id": "string", "qty": 1}
]

```

Response headers

```

content-length: 26
content-type: application/json
date: Mon, 17 Nov 2025 13:26:44 GMT
server: unicorn

```

[Download](#)

Membatalkan reservasi

POST /ohs/{sku}/release

Tampilan awal:

POST /ohs/{sku}/release Release Reservation

Parameters

Name	Description
sku <small>required</small>	sku (path)

Request body required

application/json

Example Value | Schema

```
{
  "reservation_id": "string"
}
```

Responses

Code	Description	Links
200	Successful Response	No links

Media type: application/json

Controls Accept header

Example Value | Schema

```
{
  "id": "string",
  "sku": "string",
  "qty": 1,
  "reserved": true,
  "user": "string",
  "user_id": "string",
  "with_stock": true,
  "reservations": []
}
```

Tampilan Try It Out:

POST /ohs/{sku}/release Release Reservation

Parameters

Name	Description
sku <small>required</small>	sku (path)

Request body required

application/json

Edit Value | Schema

```
{
  "reservation_id": "string"
}
```

Execute

Tampilan Edit Request Body:

Name	Description
sku * required string (path)	SKU-001

Request body required

Edit Value | Schema

```
{  
  "reservation_id": "170ba325-0387-46fe-9b2b-113c2b34fcbb"  
}
```

Tampilan Setelah Execute:

Responses

Curl

```
curl -X 'POST' '\  
http://127.0.0.1:8000/ohs/SKU-001/release'\  
-H 'Content-Type: application/json'  
-d '{  
  "reservation_id": "170ba325-0387-46fe-9b2b-113c2b34fcbb"  
}'
```

Request URL

<http://127.0.0.1:8000/ohs/SKU-001/release>

Server response

Code Details

200 Response body

```
{  
  "id": "74d31e9c-28f7-4862-90f7-09308e05cfac",  
  "sku": "SKU-001",  
  "reserved": 0,  
  "available": 100,  
  "user": "pcx",  
  "min_qty": 1,  
  "max_qty": 10,  
  "locked": false,  
  "reservations": []  
}
```

Download

Response headers

```
content-length: 163  
content-type: application/json  
date: Mon, 17 Nov 2025 13:23:30 GMT  
server: unicorn
```

Tampilan saat cek reservasi stok SKU-001: sudah kosong

Curl

```
curl -X 'GET' '\  
http://127.0.0.1:8000/ohs/SKU-001/reservations'\  
-H "accept: application/json"
```

Request URL

<http://127.0.0.1:8000/ohs/SKU-001/reservations>

Server response

Code Details

200 Response body

```
[]
```

Response headers

```
content-length: 2  
content-type: application/json  
date: Mon, 17 Nov 2025 13:24:23 GMT  
server: unicorn
```

Melihat ketersediaan stok

GET /ohs/availability/{sku}

Tampilan awal:

The screenshot shows the API documentation for the `GET /ohs/availability/{sku}` endpoint. It includes sections for **Parameters**, **Responses**, and **Example Value | Schema**. The **Parameters** section has a single required parameter `sku` of type `string` (path). The **Responses** section shows two status codes: `200` for a successful response with a media type of `application/json` containing a schema example, and `422` for a validation error.

Tampilan Try It Out:

The screenshot shows the **Try It Out** interface for the `GET /ohs/availability/{sku}` endpoint. It displays the same parameters and responses as the initial view, but includes a large blue **Execute** button at the bottom.

Tampilan Edit Request Body:

The screenshot shows the **Edit Request Body** interface for the `GET /ohs/availability/{sku}` endpoint. It shows the `sku` parameter set to `SKU-001` and a large blue **Execute** button at the bottom.

Tampilan Setelah Execute:

The screenshot shows the results of the executed request. It includes a **Curl** command, a **Request URL** (http://127.0.0.1:8000/ohs/availability/SKU-001), and a **Server response** section. The response details a `200` status code with a **Response body** containing stock availability data and **Response headers** including content-length, content-type, date, and server.

Melihat daftar low stock

GET /manager/low-stock

Tampilan awal:

The screenshot shows the API documentation for the `/manager/low-stock` endpoint. It includes sections for `Parameters` (none), `Responses`, and a `Code` section for `200` status. The `200` section details a successful response with a media type of `application/json`. An example value is provided as a JSON array:

```
[{"id": "4d34c5-28f7-4862-9ff7-d938be05cfac", "sku": "SKU-001", "on_hand": 1, "reserved": 0, "available": 1, "min_qty": 1, "low_stock": true, "reservations": []}]
```

Tampilan Try It Out:

The screenshot shows the `Try It Out` interface for the `/manager/low-stock` endpoint. It includes sections for `Parameters` (none) and a large blue `Execute` button.

Tampilan Edit Request Body: Tidak ada body

Tampilan Setelah Execute:

The screenshot shows the results of the executed request. It includes sections for `Responses`, `Curl` command, `Request URL` (`http://127.0.0.1:8000/manager/low-stock`), and `Server response`. The response code is `200` with a response body of an empty JSON array: `[]`.

Tampilan Setelah SKU-001 berada di < Min Qty:

The screenshot shows the results of the executed request after the SKU-001 quantity was reduced. The `Response body` now contains one item with `low_stock: true`:

```
[{"id": "4d34c5-28f7-4862-9ff7-d938be05cfac", "sku": "SKU-001", "on_hand": 0, "reserved": 0, "available": 0, "min_qty": 1, "low_stock": true, "reservations": []}]
```

The `Content-length` header in the `Response headers` is now `162`.

Mengatur Threshold Baru

POST
/admin/items/{sku}/threshold

Tampilan awal:

The screenshot shows the API documentation for the 'Set Threshold' endpoint. It includes fields for 'Name' (sku) and 'Request body' (with a sample value of '{}'). Below this, there's a 'Responses' section for a 200 status code, which includes a 'Media type' dropdown set to 'application/json' and a sample response schema: { "id": "string", "name": "string", "min_qty": 0, "available": 0, "now": "string", "low_stock": true, "reservations": [] }.

Tampilan Try It Out:

The screenshot shows the 'Try It Out' interface for the 'Set Threshold' endpoint. It has a 'Parameters' section with 'sku' set to 'SKU-001'. The 'Request body' section contains the value '{}'. At the bottom, there is a blue 'Execute' button.

Tampilan Edit Request Body:

The screenshot shows the 'Edit Request Body' interface. It displays a table with a single row for 'sku' (value: SKU-001). Below this is a 'Request body' section with an 'Edit Value | Schema' button. A modal window shows a JSON schema with a red underline under the 'min_qty' field: { "min_qty": 100 }.

Tampilan Setelah Execute:

```

curl -X "POST" \
  "http://127.0.0.1:8000/admin/items/SKU-001/threshold" \
  -H "Content-type: application/json" \
  -d '{
    "min_qty": 100
  }'

```

Request URL
<http://127.0.0.1:8000/admin/items/SKU-001/threshold>

Server response

Code	Details
200	Response body <pre>{ "id": "74d31e9c-28f7-4862-90f7-d93b8e05cfac", "sku": "SKU-001", "on_hand": 100, "reserved": 0, "available": 100, " uom": "pc", "min_qty": 100, "low_stock": true, "reservations": [] }</pre> <p>Copy Download</p> <p>Response headers</p> <pre>content-length: 161 content-type: application/json date: Mon, 17 Nov 2025 13:35:47 GMT server: uvicorn</pre>

Swagger UI secara otomatis menampilkan struktur request dan response, lengkap dengan tipe data, contoh input, dan dokumentasi parameter, sehingga sangat membantu dalam memastikan bahwa setiap endpoint bekerja sesuai spesifikasi.

```

[{"id": "74d31e9c-28f7-4862-90f7-d93b8e05cfac", "sku": "SKU-001", "on_hand": 100, "reserved": 0, "available": 100, " uom": "pc", "min_qty": 100, "low_stock": true, "reservations": []}, {"id": "d04b4b2e-4222-4882-89b1-04373811e033", "sku": "SKU-002", "on_hand": 200, "reserved": 0, "available": 200, " uom": "pc", "min_qty": 200, "low_stock": false, "reservations": []}]

```

[Copy](#) [Download](#)

Response headers

```
content-length: 220
content-type: application/json
date: Mon, 17 Nov 2025 13:38:46 GMT
server: uvicorn
```

Gambar 6. Tampilan Kondisi akhir Swagger Setelah Testing API di Swagger

```

1   [
2     {
3       "id": "74d31e9c-28f7-4862-90f7-d93b8e05cfac",
4       "sku": "SKU-001",
5       "on_hand": 5,
6       "reserved": 0,
7       "available": 5,
8       "uom": "pcs",
9       "min_qty": 100,
10      "low_stock": true,
11      "reservations": []
12    },
13    {
14      "id": "d04b4b3e-4222-4882-89b1-04373811e033",
15      "sku": "SKU-002",
16      "on_hand": 200,
17      "reserved": 0,
18      "available": 200,
19      "uom": "pcs",
20      "min_qty": 20,
21      "low_stock": false,
22      "reservations": []
23    }
24 ]

```

Gambar 7. Tampilan Kondisi awal Postman Setelah Testing API di Swagger

B. Pengujian Menggunakan Postman

Selain Swagger UI, pengujian juga dilakukan menggunakan Postman untuk memastikan API dapat diakses dan digunakan melalui protokol HTTP standar tanpa ketergantungan pada antarmuka FastAPI.

Membuat item baru

POST /admin/items

Body: `{
 "id": "6ea1f093-8d36-45aa-8cc6-7a7704739d0a",
 "sku": "SKU-001",
 "on_hand": 399,
 "reserved": 0,
 "available": 399,
 "uom": "pcs",
 "min_qty": 39,
 "low_stock": false,
 "reservations": []
}`

Headers: `Content-Type: application/json`

Response:

```

1   {
2     "id": "6ea1f093-8d36-45aa-8cc6-7a7704739d0a",
3     "sku": "SKU-001",
4     "on_hand": 399,
5     "reserved": 0,
6     "available": 399,
7     "uom": "pcs",
8     "min_qty": 39,
9     "low_stock": false,
10    "reservations": []
11  }

```

Melihat daftar item

GET /admin/items

```

[{"id": "79d41e9c-28f7-4862-9817-d93bde0cfac", "sku": "SKU-001", "on_hand": 100, "reserved": 0, "available": 5, "min_qty": 10, "min_qty": 100, "low_stock": true, "reservations": []}, {"id": "9e905b5e-4722-4862-89b1-04373011e033", "sku": "SKU-002", "on_hand": 200, "reserved": 0, "available": 200, "min_qty": 20, "min_qty": 20, "low_stock": false, "reservations": []}]
    
```

Melihat detail item berdasarkan SKU

GET /admin/items/{sku}

Tampilan Setelah Execute SKU-001:

```

{
  "id": "79d41e9c-28f7-4862-9817-d93bde0cfac",
  "sku": "SKU-001",
  "on_hand": 100,
  "reserved": 0,
  "available": 5,
  "min_qty": 10,
  "min_qty": 100,
  "low_stock": true,
  "reservations": []
}
    
```

Tampilan Setelah Execute SKU-003:

```

{
  "id": "6eaf1f003-8d36-45aa-8ccb-7a7704739d3a",
  "sku": "SKU-003",
  "on_hand": 300,
  "reserved": 0,
  "available": 300,
  "min_qty": 30,
  "min_qty": 30,
  "low_stock": false,
  "reservations": []
}
    
```

Tampilan Setelah Execute SKU-004:

The screenshot shows a REST API interface with a sidebar containing endpoints like 'POST CheckItem', 'POST IncreaseItem', and 'GET DecreaseItem'. The main area has tabs for 'Docs', 'Params', 'Authorization', 'Headers (8)', 'Body', 'Scripts', and 'Settings'. Under 'Params', there is a table with columns 'Key' and 'Value'. The 'Body' tab shows a JSON response with the message 'detail: "Item not found"'. The status bar at the bottom indicates '404 Not Found'.

Melakukan peningkatan stok

POST /ohs/{sku}/increase

Tampilan Setelah Execute Increase 50 Qty untuk SKU-001:

The screenshot shows a REST API interface with a sidebar containing endpoints like 'POST CheckItem', 'POST IncreaseItem', and 'GET DecreaseItem'. The main area has tabs for 'Docs', 'Params', 'Authorization', 'Headers (8)', 'Body', 'Scripts', and 'Settings'. Under 'Body', a JSON payload is shown with 'qty': 50 and 'reason': 'goods_received'. The status bar at the bottom indicates '200 OK'.

Melakukan pengurangan stok

POST /ohs/{sku}/decrease

Tampilan Setelah Execute Decrease 120 Qty untuk SKU-003:

The screenshot shows a REST API interface with a sidebar containing endpoints like 'POST CheckItem', 'POST IncreaseItem', and 'GET DecreaseItem'. The main area has tabs for 'Docs', 'Params', 'Authorization', 'Headers (8)', 'Body', 'Scripts', and 'Settings'. Under 'Body', a JSON payload is shown with 'qty': 120 and 'reason': 'order_shipped'. The status bar at the bottom indicates '200 OK'.

Melakukan reservasi stok

POST /ohs/{sku}/reserve

Tampilan Setelah Execute Reserve SKU-003:

```

POST http://127.0.0.1:8000/ohs/SKU-003/reserve
{
    "order_id": "ORD-1",
    "sku": "SKU-003",
    "qty": 20
}
200 OK
{
    "id": "639f6142-2dec-47fb-9423-b79fe635e370",
    "order_id": "ORD-1",
    "sku": "SKU-003",
    "quantity": 20,
    "uom": "pc",
    "min_qty": 30,
    "low_stock": false,
    "reservations": [
        {
            "id": "639f6142-2dec-47fb-9423-b79fe635e370",
            "order_id": "ORD-1",
            "qty": 20
        }
    ]
}

```

Melihat daftar reservasi stok

GET /ohs/{sku}/reservations

Tampilan Setelah Execute Reservations SKU-003:

```

GET http://127.0.0.1:8000/ohs/SKU-003/reservations
200 OK
{
    "id": "639f6142-2dec-47fb-9423-b79fe635e370",
    "order_id": "ORD-1",
    "sku": "SKU-003",
    "qty": 20
}

```

Tampilan Setelah Execute Reservations SKU-001:

```

GET http://127.0.0.1:8000/ohs/SKU-001/reservations
200 OK
[]

```

Membatalkan reservasi

POST /ohs/{sku}/release

Tampilan Setelah Execute Release Reservations SKU-003:

The screenshot shows a POST request to `http://127.0.0.1:8000/ohs/SKU-003/release`. The body contains the following JSON:

```
1 | { "reservation_id": "639f6e42-2dac-47fb-9423-b79fa035e378" }
2 |
3 |
4 |
5 |
6 |
7 |
8 |
9 |
10 |
11 }
```

The response status is 200 OK with a response time of 5 ms and a total size of 291 B. The response body is a JSON object with fields: `id`, `ip`, `sku`, `on_hand`, `threshold`, `available`, `uom`, `on_order`, `low_stock`, and `reservations`.

Melihat ketersediaan stok

GET /ohs/availability/{sku}

Tampilan Setelah Execute Availability SKU-003:

The screenshot shows a GET request to `http://127.0.0.1:8000/ohs/availability/SKU-003`. The response status is 200 OK with a response time of 5 ms and a total size of 219 B. The response body is a JSON array with one item:

```
1 | [
2 |   {
3 |     "sku": "SKU-003",
4 |     "on_hand": 180,
5 |     "threshold": 100,
6 |     "available": 180,
7 |     " uom": "pcs",
8 |     "on_order": 0,
9 |     "low_stock": false,
10 |     "reservations": []
11 |   }
12 | ]
```

Melihat daftar low stock

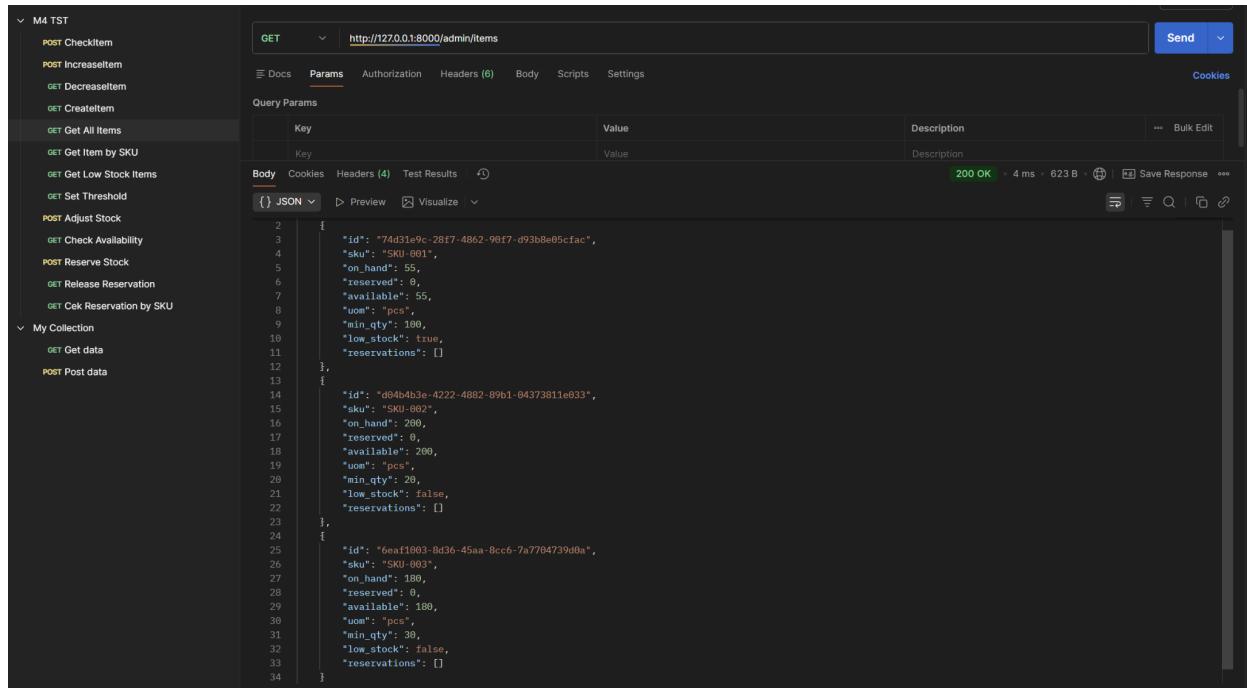
GET /manager/low-stock

Tampilan Setelah Execute Low Stock:

The screenshot shows a GET request to `http://127.0.0.1:8000/manager/low-stock`. The response status is 200 OK with a response time of 4 ms and a total size of 291 B. The response body is a JSON array with one item:

```
1 | [
2 |   {
3 |     "id": "74d319c-28f7-4862-98f7-d93b8e95cfac",
4 |     "ip": "127.0.0.1:8003",
5 |     "on_hand": 9001,
6 |     "reserved": 0,
7 |     "available": 55,
8 |     " uom": "pcs",
9 |     "min_stk": 100,
10 |     "low_stock": true,
11 |     "reservations": []
12 |   }
13 | ]
```

Postman juga digunakan untuk menguji error handling, seperti mencoba reservasi di luar batas stok, menghapus reservasi yang tidak ada, atau melakukan decrease stok melebihi available.



The screenshot shows the Postman application interface. On the left, there's a sidebar with a tree view of API endpoints under 'M4 TST' and 'My Collection'. The main area shows a 'GET' request to 'http://127.0.0.1:8000/admin/items'. The 'Params' tab is selected, showing a table with 'Key' and 'Value' columns. The 'Body' tab is selected, showing a JSON response with three items. The response body is as follows:

```
2   [
3     {
4       "id": "74d31e9c-28f7-4862-90f7-d93b8e05cfac",
5       "sku": "SKU-001",
6       "on_hand": 55,
7       "reserved": 0,
8       "available": 55,
9       "uom": "pcs",
10      "min_qty": 100,
11      "low_stock": true,
12      "reservations": []
13    },
14    {
15      "id": "d84b4b3e-4222-4882-89b1-04373811e033",
16      "sku": "SKU-002",
17      "on_hand": 280,
18      "reserved": 0,
19      "available": 280,
20      "uom": "pcs",
21      "min_qty": 20,
22      "low_stock": false,
23      "reservations": []
24    },
25    {
26      "id": "6eaf1003-8d36-45aa-8cc6-7a7704739d0a",
27      "sku": "SKU-003",
28      "on_hand": 180,
29      "reserved": 0,
30      "available": 180,
31      "uom": "pcs",
32      "min_qty": 30,
33      "low_stock": false,
34      "reservations": []
35    }
36  ]
```

Gambar 8. Tampilan Kondisi akhir Postman Setelah Testing API di Postman

The screenshot shows the Postman interface for a GET request to the endpoint `/admin/items`. The request parameters are listed as "No parameters". Below the request URL, the response code is 200. The response body is a JSON array containing three items:

```

[{"id": "d04b4b3e-4222-4882-89b1-04373811e033", "sku": "SKU-001", "on_hand": 25, "reserved": 0, "available": 55, "uom": "pcs", "min_qty": 100, "low_stock": true, "reservations": []}, {"id": "604b4b3e-4222-4882-89b1-04373811e033", "sku": "SKU-002", "on_hand": 200, "reserved": 0, "available": 200, "uom": "pcs", "min_qty": 20, "low_stock": false, "reservations": []}, {"id": "604b4b3e-4222-4882-89b1-04373811e033", "sku": "SKU-003", "on_hand": 180, "reserved": 0, "available": 180, "uom": "pcs", "min_qty": 30, "low_stock": false, "reservations": []}]

```

Gambar 9. Tampilan Kondisi akhir Swagger Setelah Testing API di Postman

Dari seluruh pengujian manual menggunakan Swagger UI dan Postman, seluruh endpoint Inventory Control berjalan sesuai ekspektasi. Setiap operasi seperti penambahan stok, pengurangan stok, reservasi, pelepasan reservasi, dan pengecekan low stock berhasil menghasilkan respons yang konsisten dengan logika domain yang telah diimplementasikan. Selain itu, komponen `InventoryRepositoryInMemory` yang digunakan pada tahap ini juga berfungsi dengan baik, dimana setiap perubahan data yang dilakukan melalui Swagger UI langsung tercermin ketika diuji ulang melalui Postman, dan sebaliknya. Hal ini menunjukkan bahwa repository in-memory mampu menyimpan state aplikasi secara konsisten selama server berjalan, sehingga seluruh proses pengujian menunjukkan sinkronisasi data yang benar antara kedua alat tersebut. Secara keseluruhan, hasil ini menandakan bahwa implementasi API dan mekanisme penyimpanan sementara telah bekerja secara stabil dan sesuai dengan desain modul Inventory Control.

```
(.venv)
Ivana@LAPTOP-06GCRACU MINGW64 ~/OneDrive/Documents/II3160_TB_18223126_WMS (main)
$ uvicorn src.main:app --reload
INFO:     Application startup complete.
INFO: 127.0.0.1:57604 - "GET / HTTP/1.1" 307 Temporary Redirect
INFO: 127.0.0.1:57604 - "GET /docs HTTP/1.1" 200 OK
INFO: 127.0.0.1:57604 - "GET /openapi.json HTTP/1.1" 200 OK
INFO: 127.0.0.1:65197 - "POST /admin/items HTTP/1.1" 200 OK
INFO: 127.0.0.1:59291 - "POST /admin/items HTTP/1.1" 200 OK
INFO: 127.0.0.1:53772 - "GET /admin/items HTTP/1.1" 200 OK
INFO: 127.0.0.1:55730 - "POST /admin/items HTTP/1.1" 400 Bad Request
INFO: 127.0.0.1:52722 - "GET /admin/items/SKU-001 HTTP/1.1" 200 OK
INFO: 127.0.0.1:60875 - "POST /ohs/SKU-001/increase HTTP/1.1" 200 OK
INFO: 127.0.0.1:56282 - "POST /ohs/SKU-001/decrease HTTP/1.1" 200 OK
INFO: 127.0.0.1:59667 - "GET /ohs/SKU-001/reservations HTTP/1.1" 200 OK
INFO: 127.0.0.1:63732 - "POST /ohs/SKU-001/reserve HTTP/1.1" 200 OK
INFO: 127.0.0.1:62276 - "GET /ohs/SKU-001/reservations HTTP/1.1" 200 OK
INFO: 127.0.0.1:59039 - "POST /ohs/SKU-001/release HTTP/1.1" 200 OK
INFO: 127.0.0.1:59153 - "GET /ohs/SKU-001/reservations HTTP/1.1" 200 OK
INFO: 127.0.0.1:61215 - "GET /ohs/availability/SKU-001 HTTP/1.1" 200 OK
INFO: 127.0.0.1:59791 - "GET /manager/low-stock HTTP/1.1" 200 OK
INFO: 127.0.0.1:58305 - "POST /ohs/SKU-001/decrease HTTP/1.1" 400 Bad Request
INFO: 127.0.0.1:58147 - "POST /ohs/SKU-001/decrease HTTP/1.1" 200 OK
INFO: 127.0.0.1:61984 - "GET /manager/low-stock HTTP/1.1" 200 OK
INFO: 127.0.0.1:53482 - "POST /ohs/SKU-001/increase HTTP/1.1" 200 OK
INFO: 127.0.0.1:59225 - "GET /manager/low-stock HTTP/1.1" 200 OK
INFO: 127.0.0.1:59225 - "GET /manager/low-stock HTTP/1.1" 200 OK
INFO: 127.0.0.1:51497 - "POST /ohs/SKU-001/decrease HTTP/1.1" 400 Bad Request
INFO: 127.0.0.1:61339 - "POST /ohs/SKU-001/decrease HTTP/1.1" 200 OK
INFO: 127.0.0.1:59182 - "GET /manager/low-stock HTTP/1.1" 200 OK
INFO: 127.0.0.1:54170 - "POST /admin/items/SKU-001/threshold HTTP/1.1" 200 OK
INFO: 127.0.0.1:64495 - "GET /admin/items HTTP/1.1" 200 OK
INFO: 127.0.0.1:64440 - "GET /admin/items HTTP/1.1" 200 OK
INFO: 127.0.0.1:64506 - "POST /admin/items HTTP/1.1" 200 OK
INFO: 127.0.0.1:61248 - "GET /admin/items HTTP/1.1" 200 OK
INFO: 127.0.0.1:55866 - "GET /admin/items HTTP/1.1" 200 OK
INFO: 127.0.0.1:54150 - "GET /admin/items/SKU-003 HTTP/1.1" 200 OK
INFO: 127.0.0.1:49679 - "GET /admin/items/SKU-001 HTTP/1.1" 200 OK
INFO: 127.0.0.1:49699 - "GET /admin/items/SKU-004 HTTP/1.1" 404 Not Found
INFO: 127.0.0.1:65038 - "POST /ohs/SKU-001/increase HTTP/1.1" 200 OK
INFO: 127.0.0.1:65079 - "POST /ohs/SKU-003/decrease HTTP/1.1" 200 OK
INFO: 127.0.0.1:57299 - "POST /ohs/SKU-003/reserve HTTP/1.1" 200 OK
INFO: 127.0.0.1:57322 - "GET /ohs/SKU-003/reservations HTTP/1.1" 200 OK
INFO: 127.0.0.1:57339 - "GET /ohs/SKU-001/reservations HTTP/1.1" 200 OK
INFO: 127.0.0.1:51995 - "GET /ohs/SKU-003/reservations HTTP/1.1" 200 OK
INFO: 127.0.0.1:52005 - "POST /ohs/SKU-001/release HTTP/1.1" 400 Bad Request
INFO: 127.0.0.1:52012 - "GET /ohs/SKU-003/reservations HTTP/1.1" 200 OK
INFO: 127.0.0.1:52026 - "POST /ohs/SKU-001/release HTTP/1.1" 400 Bad Request
INFO: 127.0.0.1:52033 - "POST /ohs/SKU-003/release HTTP/1.1" 200 OK
INFO: 127.0.0.1:50833 - "GET /ohs/availability/SKU-003 HTTP/1.1" 200 OK
INFO: 127.0.0.1:50848 - "GET /manager/low-stock HTTP/1.1" 200 OK
INFO: 127.0.0.1:50687 - "GET /admin/items HTTP/1.1" 200 OK
INFO: 127.0.0.1:50730 - "GET /admin/items HTTP/1.1" 200 OK
```

Gambar 10. Tampilan Terminal VSCode saat Melakukan API Testing pada Swagger dan Postman

C. Pengujian Menggunakan pytest

Selain pengujian manual menggunakan Swagger UI dan Postman, sistem Inventory Control juga diuji menggunakan pengujian otomatis berbasis pytest. Tujuan dari pengujian otomatis ini adalah memastikan bahwa logika domain maupun endpoint API berfungsi secara konsisten, dapat diuji ulang, dan tidak mudah mengalami regresi apabila dilakukan perubahan kode di masa depan. Pendekatan ini memberikan jaminan kualitas yang lebih kuat dibandingkan pengujian manual semata.

Pengujian otomatis dilakukan pada dua level, yaitu pengujian domain dan pengujian API. Pengujian domain berfokus pada perilaku kelas `InventoryItem` sebagai Aggregate Root beserta Value Object dan Entity yang menyertainya. Beberapa skenario yang diuji meliputi penambahan stok (`increase`), pengurangan stok (`decrease`), pembuatan reservasi (`reserve`), pelepasan reservasi (`release`), serta pengecekan kondisi *low stock*. Setiap pengujian memastikan bahwa invariant domain (seperti `reserved` tidak boleh melebihi `on_hand`) tetap terjaga dan setiap operasi menghasilkan perubahan state yang benar.

Sementara itu, pengujian API dilakukan menggunakan `TestClient` dari FastAPI, yang mensimulasikan request HTTP tanpa perlu menjalankan server Unicorn. Pengujian ini meliputi pembuatan item baru melalui endpoint admin, pemanggilan operasi increase dan decrease, reservasi dan pelepasan reservasi, serta pengecekan ketersediaan stok melalui endpoint OHS. Dengan pendekatan ini, API dapat diuji secara end-to-end, mulai dari pemanggilan endpoint hingga respons yang dihasilkan, tanpa melibatkan infrastruktur eksternal. Seluruh pengujian otomatis dijalankan melalui perintah:

```
pytest -v
```

```

Ivana@LAPTOP-06GCRACU MINGW64 ~/OneDrive/Documents/II3160_TB_18223126_WMS (main)
● $ git checkout feature/tests
Switched to branch 'feature/tests'
Your branch is up to date with 'origin/feature/tests'.
(.venv)
Ivana@LAPTOP-06GCRACU MINGW64 ~/OneDrive/Documents/II3160_TB_18223126_WMS (feature/tests)
● $ pytest -v
=====
platform win32 -- Python 3.13.5, pytest-9.0.1, pluggy-1.6.0 -- c:\Users\Ivana\OneDrive\Documents\II3160_TB_18223126_WMS\.venv\Scripts\python.exe
cachedir: .pytest_cache
rootdir: C:\Users\Ivana\OneDrive\Documents\II3160_TB_18223126_WMS
plugins: asyncio-4.11.0
collected 9 items

tests/test_api_inventory.py::test_create_item_success PASSED [ 11%]
tests/test_api_inventory.py::test_get_item_after_create PASSED [ 22%]
tests/test_api_inventory.py::test_increase_and_decrease_stock_via_api PASSED [ 33%]
tests/test_api_inventory.py::test_availability_endpoint PASSED [ 44%]
tests/test_domain_inventory.py::test_increase_stock_adds_on_hand PASSED [ 55%]
tests/test_domain_inventory.py::test_reserve_reduces_available_and_increases_reserved PASSED [ 66%]
tests/test_domain_inventory.py::test_reserve_raises_when_not_enough_available PASSED [ 77%]
tests/test_domain_inventory.py::test_decrease_reduces_on_hand_and_available PASSED [ 88%]
tests/test_domain_inventory.py::test_low_stock_flag_works PASSED [100%]

===== warnings summary =====
src\schemas\inventory.py:33
  C:\Users\Ivana\OneDrive\Documents\II3160_TB_18223126_WMS\src\schemas\inventory.py:33: PydanticDeprecatedSince20: Using extra keyword arguments on `Field` is deprecated and will be removed. Use `json_schema_extra` instead. (Extra keys: 'example'). Deprecated in Pydantic V2.0 to be removed in V3.0. See Pydantic V2 Migration Guide at https://errors.pydantic.dev/2.12/migration/
    sku: str = Field(..., example="SKU-001")

tests/test_api_inventory.py::test_increase_and_decrease_stock_via_api
tests/test_api_inventory.py::test_increase_and_decrease_stock_via_api
tests/test_api_inventory.py::test_availability_endpoint
tests/test_domain_inventory.py::test_increase_stock_adds_on_hand
tests/test_domain_inventory.py::test_reserve_reduces_available_and_increases_reserved
tests/test_domain_inventory.py::test_decrease_reduces_on_hand_and_available
tests/test_domain_inventory.py::test_low_stock_flag_works
  <string>:6: DeprecationWarning: datetime.utcnow() is deprecated and scheduled for removal in a future version. Use timezone-aware objects to represent dates in UTC: datetime.datetime.now(datetime.UTC).

-- Docs: https://docs.pytest.org/en/stable/how-to/capture-warnings.html
===== 9 passed, 8 warnings in 0.86s =====
(.venv)
Ivana@LAPTOP-06GCRACU MINGW64 ~/OneDrive/Documents/II3160_TB_18223126_WMS (feature/tests)

```

Gambar 11. Tampilan Terminal VSCode saat Melakukan API Testing dengan Pytest (pada branch feature/tests)

Pengujian otomatis menggunakan pytest dilakukan untuk memverifikasi bahwa logika domain dan seluruh endpoint API berfungsi sesuai dengan aturan bisnis yang telah diimplementasikan. Dari total sembilan test case yang dijalankan, meliputi pengujian penambahan stok, pengurangan stok, reservasi, pelepasan reservasi, pengecekan ketersediaan, dan validasi low stock, seluruhnya berhasil lulus dengan status 9 passed. Hasil ini menunjukkan bahwa perilaku Aggregate `InventoryItem` dan alur kerja pada Application Layer berjalan konsisten dan tidak menghasilkan error pada berbagai skenario operasi inventori.

Selama pengujian, beberapa peringatan (warning) muncul terkait perubahan pada Pydantic dan penggunaan `datetime.utcnow()`, namun hal tersebut tidak mempengaruhi validitas hasil pengujian maupun fungsi sistem secara keseluruhan. Secara keseluruhan, pengujian otomatis ini membuktikan bahwa arsitektur dan implementasi modul Inventory Control stabil, konsisten, serta mampu mempertahankan integritas data selama berbagai operasi pengelolaan stok dilakukan.

KESIMPULAN

Implementasi modul Inventory Control berhasil merealisasikan rancangan domain yang telah disusun sebelumnya ke dalam sistem yang berjalan secara konsisten dan fungsional. Aggregate `InventoryItem` beserta Value Object dan Entity pendukungnya mampu menangani seluruh aturan bisnis inti, seperti penambahan stok, pengurangan stok, reservasi, pelepasan reservasi, hingga penyesuaian stok. Application Layer menyediakan koordinasi use case yang jelas, sementara API FastAPI berfungsi sebagai antarmuka operasional yang dapat diakses oleh aktor seperti admin gudang, sistem Order/Outbound/Inbound melalui OHS, serta manajer untuk kebutuhan monitoring. Selama pengujian menggunakan Swagger UI dan Postman, seluruh endpoint memberikan respons yang stabil dan saling sinkron, menandakan bahwa `InventoryRepositoryInMemory` bekerja dengan baik dalam menyimpan state aplikasi selama server berjalan.

Pengujian otomatis menggunakan pytest turut memperkuat stabilitas implementasi, di mana seluruh sembilan test case pada level domain dan API berhasil dijalankan tanpa error. Hasil ini menunjukkan bahwa invariant domain terjaga, perilaku operasi inventori sesuai dengan ekspektasi, dan alur request-response API telah bekerja dengan benar. Meskipun beberapa peringatan muncul terkait deprecations pada library tertentu, hal tersebut tidak mempengaruhi kebenaran logika maupun performa sistem secara keseluruhan. Ini menegaskan bahwa modul telah siap digunakan dan dapat menjadi pondasi yang kuat untuk tahap pengembangan berikutnya.

Secara keseluruhan, implementasi Inventory Control berhasil memenuhi tujuan utama yaitu menyediakan layanan pengelolaan stok yang stabil, terstruktur, dan konsisten dengan prinsip Domain-Driven Design. Sistem telah menyediakan fitur-fitur inti pengelolaan stok yang dapat diuji dengan berbagai metode dan bekerja secara reliabel. Dengan fondasi implementasi yang kokoh ini, modul siap untuk diperluas ke tahap selanjutnya seperti integrasi database, event-driven workflow, maupun pengembangan antarmuka pengguna.

Lampiran:

Link repository: https://github.com/meerancor33/II3160_TB_18223126_WMS.git

Link video demo: <https://youtu.be/RSTUMIqXdmU>