# Diamonds Are Forever

## Inventory Management API

White diamonds · Colored diamonds · Gemstones

## Project Overview

- Web application for managing precious stones inventory
- CRUD API with authentication
- Centralized inventory system
- Full traceability of precious stones
- Scalable and production-ready API

## Main Features

- Browse inventory (public access)
- Create, update, delete inventory items
- Track item lifecycle (purchase, transfer, sale)
- Session-based authentication for employees

## Authentication

- Employee sign-in using session cookies
- Protected endpoints for write operations
- Public read-only access for inventory browsing

## Deployment

- Dockerized application
- Hosted on a cloud virtual machine
- Domain + DNS configuration
- HTTPS via reverse proxy

## API Design

- CRUD
- JSON request/response format
- CRUD-based operations
- Clear HTTP status codes

## GET Endpoints

| Endpoint | Description | Status codes |
|---|---|---|
| `GET /items` | List all inventory items | `200` |
| `GET /items/lifecycle/{id}` | Get lifecycle (actions) of an item | `200`, `404` |
| `GET /profile` | Get current user profile | `200`, `401` |
| `GET /white-diamonds/{id}` | Get white diamond details | `200`, `404` |
| `GET /colored-diamonds/{id}` | Get colored diamond details | `200`, `404` |
| `GET /colored-gemstones/{id}` | Get colored gemstone details | `200`, `404` |

## GET with query parameter

| Parameter | Type | Example | Description |
|---|---|---|---|
| `isAvailable` | boolean | `?isAvailable=True` | available ? |
| `type` | string | `?type=white%20diamond` | Filters by item type |

## POST Endpoints

| Endpoint | Description | Status codes |
|---|---|---|
| `POST /sign-in` | Sign in as an employee | `204`, `400`, `401` |
| `POST /sign-out` | Sign out (invalidate session) | `204` |
| `POST /white-diamonds` | Create a new white diamond | `201`, `400`, `401` |
| `POST /colored-diamonds` | Create a new colored diamond | `201`, `400`, `401` |
| `POST /colored-gemstones` | Create a new colored gemstone | `201`, `400`, `401` |
| `POST /actions` | Create a new lifecycle action | `201`, `400`, `401`, `409` |

## PUT Endpoints

| Endpoint | Description | Status codes |
|---|---|---|
| `PUT /white-diamonds/{id}` | Update white diamond information | `200`, `400`, `401`, `404` |
| `PUT /colored-diamonds/{id}` | Update colored diamond information | `200`, `400`, `401`, `404` |
| `PUT /colored-gemstones/{id}` | Update colored gemstone information | `200`, `400`, `401`, `404` |

## DELETE Endpoints

| Endpoint | Description | Status codes |
|---|---|---|
| `DELETE /items/{id}` | Delete item and all related actions | `200`, `401`, `404` |
| `DELETE /actions/{id}` | Delete most recent action of an item | `200`, `401`, `404`, `409` |

## Cache strategy (1)

### Actions cache

The idea of actions cache is very simple:

- when user requests GET on /items/lifecycle/{id} we return him lifecycle (set of actions) of some item
- when action is created/deleted for certain item, the entry in hash map for this item becomes invalidated therefore needs to be re-fetched
- also an entry in actions cache becomes invalidated if corresponding item gets deleted

## Cache strategy (1)

### Items cache

Idea: items cache becomes invalidated in 3 cases:

- when we update item
- when we create item
- when we delete item

The rest of the time it is considered valid.
On database level all the stones are descendants of item, we can use this fact to synchronize cache across this 3 domains.

## Demo

## Use Cases

- Inventory viewing for visitors
- Inventory management by employees
- Lifecycle auditing and traceability

## Further Improvements

- Stronger authentication and role-based access
- Improved validation and business rules
- OpenAPI / Swagger documentation

## Thank you!