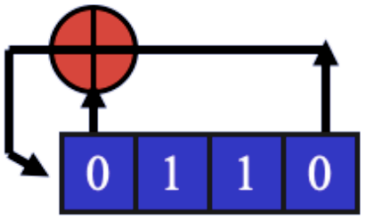
Assignment Task 1: Linear Feedback Shift Register (LFSR) Implementation

1. Overview



t	R_3	R_2	R_1	R_0
0	**0**	**1**	**1**	**0**
1	0	0	1	1
2	1	0	0	1
3	0	1	0	0
4	0	0	1	0
5	0	0	0	1
6	1	0	0	0
7	1	1	0	0

t	R_3	R_2	R_1	R_0
8	1	1	1	0
9	1	1	1	1
10	0	1	1	1
11	1	0	1	1
12	0	1	0	1
13	1	0	1	0
14	1	1	0	1
15	**0**	**1**	**1**	**0**

LFSRs are used in stream ciphers. This task involves implementing:
a) A **basic LFSR** with a fixed configuration.
b) A **general LFSR** that allows customization.

If unsure, start with (a) before moving to (b). In future tasks, you'll implement three LFSRs—either separately or using the general class.

2. Basic LFSR Implementation

Your LFSR should:

- Set the **initial state** (e.g., `0110`).
- Retrieve the **current state**.
- Generate the **next stream bit**, updating the state accordingly.

Steps:

1. Implement an LFSR with a hardwired feedback function.
2. Write a program that:
    - Initializes the state to `0110`.
    - Prints the state and the next stream bit **20 times**.
    - Confirms the output matches expected results.

3. General (Reconfigurable) LFSR

Implement an LFSR class that allows:

- Setting/getting **register size** (e.g., 4 bits).
- Setting/getting **current state**.
- Defining the **tap sequence** (XOR indices for feedback).
- Resetting the register.
- Retrieving the **next stream bit** while updating the state.

Your program should:

- Instantiate this general LFSR to match the **basic LFSR** case.
- Verify it produces the same output.

Assignment Task 2: Stock Warehouse

1. Overview

Let's say there is a distribution warehouse that conducts transactions to sell goods in large quantities to retailers. You are required to create a simple API system that performs functions for ordering and selling goods in this warehouse. For audit purposes, you are also required to make a report on changes in the stock of existing items.

2. API Specification

You must follow the listed specifications to create this API.

A. Base Model

Description:
In every model there are mandatory attributes to identify records. Any deletion operation using soft delete.
Attribute :

1. created_at
2. updated_at
3. Is_deleted

B. Modules

1. Items

Description :
Stored items in the warehouse. Quantity and balance value is 0 by default, only can be changed by purchase and sell.

Attribute :

1. code
2. name
3. unit

4. description
5. stock
6. balance

Mandatory API:
`[GET] /items/ :`
get all items

`[GET] /items/{code} :`
get an item with corresponding code on params

`[POST] /items/ :`
create an item

`[PUT] /items/{code} :`
update an item

`[DEL] /items/{code} :`
soft delete item

2. Purchases

Description :
Module for adding or replenishing stock of items in the warehouse. It has a header to store informational data and details for any bought items in that purchase.

a. Header
Description :
Informational data for a purchase

Attribute :
1. code
2. date
3. description

Mandatory API:
`[GET] /purchase/ :`
get all purchases

`[GET] /purchase/{code} :`
get a purchase with corresponding code on params

`[POST] /purchase/ :`
create a purchase

`[PUT] /purchase/{code} :`
update a purchase

`[DEL] /purchase/{code} :`
soft delete purchase

b. Detail
Description :
Contain all bought items in 1 purchasing code (many to 1 relation with header)

Attribute :
1. item_code
2. quantity
3. unit_price
4. header_code

Mandatory API:
`[GET] /purchase/{header_code}/details :`
get all purchases detail with corresponding header code on params

`[POST] /purchase/{header_code}/details :`
create purchases detail with corresponding header code on params. Should add item stock and balance based on quantity and unit_price when created

3. Sells

Description :
Module for selling stock of items in the warehouse. It has a header to store informational data and details for any sold items in that sells.

a. Header
Description :
Informational data for a sell

Attribute :
1. code
2. date
3. description

Mandatory API:
`[GET] /sell/ :`
get all sells

`[GET] /sell/{code} :`
get a sell with corresponding code on params

`[POST] /sell/ :`
create a sell

`[PUT] /sell/{code} :`
update a sell

`[DEL] /sell/{code} :`
soft delete sell

b. Detail
Description :
Contain all bought items in 1 selling code (many to 1 relation with header)

Attribute :
1. item_code
2. quantity
3. header_code

Mandatory API:
`[GET] /sell/{header_code}/details :`
get all sells detail with corresponding header code on params

`[POST] /sell/{header_code}/details :`
create sell detail with corresponding header code on params. Should decrease item stock and balance based on quantity and purchasing stock that happened. You can see the example in the report.

C. Reporting

Make a report on changes in the stock of existing items on certain date periods. Create an API endpoint to generate reports in JSON or PDF.

`[GET]/report/{item_code}/?start_date=yyyy-mm-dd&end_date=yyyy-mm-dd:`
Get a report with corresponding item code.

Query params:
start_date: contain string of start date of the report
start_date: contain string of end date of the report

For example we have this book item :

```
{
	“code” : “I-001”,
	“name”: “History Book”,
	“unit”: “Pcs”,
	“description”: “Books that tells history of the ancient”,
	“stock” : 5,
	“balance: 250000
}
```

With these purchasing:

```
{
	“code” : “P-001”,
	“date”: “2025-01-01”,
	“description”: “Buy history books”,
	“details” : [
		{
			“item_code”:”I-001”,
			“quantity”: 10,
			“unit_price” : 60000,
			“deader_code”: “P-001”
		},
	]
},
{
	“code” : “P-002”,
	“date”: “2025-02-01”,
	“description”: “Restock history books”,
	“details” : [
		{
			“item_code”:”I-001”,
			“quantity”: 10,
			“unit_price” : 50000,
			“deader_code”: “P-002”
		},
	]
}
```

And we have one transaction to sell the book

```
{
	“code” : “S-001”,
	“date”: “2025-03-01”,
	“description”: “Sell history books to library”,
	“details” : [
		{
			“item_code”:”I-001”,
			“quantity”: 15,
			“header_code”: “S-001”
		},
	]
}
```

The generated report in `[GET]/report/I-001/?start_date=2025-01-01&end_date=2025-03-31` should be like this:

Stock Report

Items code : I-001
Name : History Book
Unit : Pcs

no	Date	Description	Code	In			Out			Stock		
				qty	price	total	qty	price	total	qty	price	total
1	01-01-2025	Buy history books	P-001	10	60000	600000	0	0	0	10	60000	600000
Balance										10	600000	
2	01-02-2025	Restock history books	P-002	10	50000	500000	0	0	0	10	60000	600000
										10	50000	500000
Balance										20	1100000	
3	01-03-2025	Sell history books to library	S-001	0	0	0	10	60000	600000	0	0	0
										10	50000	500000
Balance										10	500000	
4	01-03-2025	Sell history books to library	S-001	0	0	0	5	50000	500000	0	0	0
										5	50000	250000
Balance										5	250000	
Summary				20			15			5	250000	

In JSON you can see the example here:
https://pastebin.com/AW3FSSq8

Notes :
Sales decrease item stock and balance based on stock from purchasing items.

D. Tech Stack

Mandatory library:

1. Django 5.1.3 or later
2. Django-rest-framework 3.15.2 or later

You can use any other libraries or packages to develop faster and optimize your code.

Submitting Your Result

Send your code with Git repository. For the directory structure you can make it separated for each assignment. You can separate it like this :

```
├── README.md
├── Assignment 1/
│   └── # Your code goes here
└── Assignment 2/
    └── # Django project goes here
```

Please comment on your code to explain the logic process that is happening within your code!

Once it done, please submit your GitHub repository link through the following form: https://forms.gle/D653UiybZbPHxq6R6

Good Luck on your test! Thank you.