LinkedIn Smart Contract: Attractive Summary

This document provides an attractive and structured summary of the **LinkedIn Smart Contract**, a minimal example of an on-chain posting system built with Solidity. The contract demonstrates core concepts like structs, mappings, and access control.

1. Contract Overview

Aspect	Description
Contract Name	`LinkedIn`
Purpose	Implements a simple, on-chain social media posting system.
Technology	Solidity (Smart Contract Language)
License	MIT (as per `// SPDX-License-Identifier: MIT`)

2. Key Features

The contract's functionality is centered around user posts and basic interaction.

Feature	Details
Post Length Control	The contract `owner` can set the maximum character length for a post. The default is **280 characters**.
Post Structure	Each post is stored as a `LinkedInPost` struct containing all necessary metadata.
User Interaction	Any user can **like** or **unlike** a post, regardless of who created it.
Data Storage	Posts are stored per user address using a `mapping`.

3. Contract Structure

Data Types (`struct` and `mapping`)

The core data structure is the `LinkedInPost` struct, and all posts are stored in a `mapping` .

Element	Type	Description
`MaxLength`	`uint`	The maximum allowed characters for a post. Initialized to 280.
`owner`	`address`	The address that deployed the contract, with special permissions.
`LinkedInPost`	`struct`	Defines the content of a single post: `id` , `author` , `content` , `timestamp` , `likes` .
`posts`	`mapping`	Stores all posts: `mapping(address => LinkedInPost[])` . The key is the user's address, and the value is an array of their posts.

Functions (Methods)

The contract exposes several public and external functions for interaction.

Function	Visibility	Description
`constructor()`	`public`	Sets the contract deployer (`msg.sender`) as the `owner` .
`changePostLength(uint _newLength)`	`public`	Allows the `owner` to update the `MaxLength` for posts.
`createPost(string memory _msg)`	`public`	Creates a new post, subject to the `MaxLength` check.
`likePost(address _author, uint _index)`	`external`	Increments the like count for a specific post by a specific author.

`unlikePost(address _author, uint _index)`	`external`	Decrements the like count for a specific post (must be > 0).
`getOnePost(address _author, uint _index)`	`public view`	Retrieves a single post by author address and index.
`getAllPosts(address _author)`	`public view`	Retrieves all posts made by a specific author address.

4. Full Solidity Code

Plain Text

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract LinkedIn {
    uint public MaxLength = 280;
    address public owner;

    struct LinkedInPost {
        uint id;
        address author;
        string content;
        uint timestamp;
        uint likes;
    }

    mapping(address => LinkedInPost[]) public posts;

    constructor() {
        owner = msg.sender;
    }

    modifier onlyOwner() {
        require(msg.sender == owner, "Only owner can change length");
        _;
    }

    function changePostLength(uint _newLength) public onlyOwner {
        MaxLength = _newLength;
    }

    function createPost(string memory _msg) public {
        require(bytes(_msg).length <= MaxLength, "Max characters reached");
```

```
        LinkedInPost memory newPost = LinkedInPost({
            id: posts[msg.sender].length,
            author: msg.sender,
            content: _msg,
            timestamp: block.timestamp,
            likes: 0
        });

        posts[msg.sender].push(newPost);
    }

    function likePost(address _author, uint _index) external {
        require(_index < posts[_author].length, "Post does not exist");
        require(posts[_author][_index].id == _index, "Post id mismatch");
        posts[_author][_index].likes++;
    }

    function unlikePost(address _author, uint _index) external {
        require(_index < posts[_author].length, "Post does not exist");
        require(posts[_author][_index].id == _index, "Post id mismatch");
        require(posts[_author][_index].likes > 0, "No likes to remove");
        posts[_author][_index].likes--;
    }

    function getOnePost(address _author, uint _index) public view returns
(LinkedInPost memory) {
        require(_index < posts[_author].length, "Post does not exist");
        return posts[_author][_index];
    }

    function getAllPosts(address _author) public view returns
(LinkedInPost[] memory) {
        return posts[_author];
    }
}
```