```solidity
// SPDX-License-Identifier: MIT
// OpenZeppelin Contracts (last updated v4.7.0) (proxy/beacon/BeaconProxy.sol)

pragma solidity ^0.8.0;

import "./IBeacon.sol";
import "../Proxy.sol";
import "../ERC1967/ERC1967Upgrade.sol";

/**
 * @dev This contract implements a proxy that gets the implementation address for each call from an {UpgradeableBeacon}.
 *
 * The beacon address is stored in storage slot `uint256(keccak256('eip1967.proxy.beacon')) - 1`, so that it doesn't
 * conflict with the storage layout of the implementation behind the proxy.
 *
 * _Available since v3.4._
 */
contract BeaconProxy is Proxy, ERC1967Upgrade {
    /**
     * @dev Initializes the proxy with `beacon`.
     *
     * If `data` is nonempty, it's used as data in a delegate call to the implementation returned by the beacon. This
     * will typically be an encoded function call, and allows initializing the storage of the proxy like a Solidity
     * constructor.
     *
     * Requirements:
     *
     * - `beacon` must be a contract with the interface {IBeacon}.
     */
    constructor(address beacon, bytes memory data) payable {
        _upgradeBeaconToAndCall(beacon, data, false);
    }

    /**
     * @dev Returns the current beacon address.
     */
    function _beacon() internal view virtual returns (address) {
        return _getBeacon();
    }

    /**
     * @dev Returns the current implementation address of the associated beacon.
     */
    function _implementation() internal view virtual override returns (address) {
        return IBeacon(_getBeacon()).implementation();
    }

    /**
     * @dev Changes the proxy to use a new beacon. Deprecated: see {_upgradeBeaconToAndCall}.
     *
     * If `data` is nonempty, it's used as data in a delegate call to the implementation returned by the beacon.
     *
     * Requirements:
     *
     * - `beacon` must be a contract.
     * - The implementation returned by `beacon` must be a contract.
     */
    function _setBeacon(address beacon, bytes memory data) internal virtual {
        _upgradeBeaconToAndCall(beacon, data, false);
    }
}
```