WHITE PAPER

OVER STONES
NFTs COLLECTION
https://overstones.org

It seems very necessary Necessity of transforming ancient objects and historical artifacts into NFT.
Many historical artifacts are fragile and susceptible to damage from environmental factors, such as light, temperature,humidity and battles.The Nazis ransacked Europe's museums and galleries during World War II. In Libya, historical sites were vandalised, looted and destroyed. The Taliban used high explosives to eradicate virtually every trace of the statues of Buddha at Bamiyan in Afghanistan.
Today, the only truly safe place to preserve our cultural heritage is blockchain. To make our shared culture preservable, we need to make it shareable, and NFTs give us this ability to introduce future generations to the civilization of our ancestors.

Overstones is a NFTcollection of 200 ancient stones (made by man) dating back to the millennium BC each of which is named after one of the cryptocurrencies.
This collection shows a combination of human civilizations in the distant millennia and the blockchain age Which are combined in the form of 200 unique NFTs
The ingredients of these stones are a combination of gold, Silver and Kaolinite And each of these stones are unique and different from each other in terms of weight and volume and engraved cuneiform lines.

overstones, has chosen Polygon chain for its collection because Polygon ,the most widely used Ethereum scaling ecosystem that offers EVM compatibility and an ultimate user experience with fast transactions at near-zero gas fees and adopted by the biggest projects and Community support.
Polygon is the most proven scaling solution in Web3.
Deployment onto EVM without changes in code ,Allows developers to focus on improving code (in Polygon zkEVM) rather than re-writing it.Ethereum security inherited in L2 with the additional benefit of L2 batching for scaling.

Overstones has chosen the ERC1155 standard and multiples for its collection because the monopolization of NFTs historical artifacts, which are considered a world heritage, by one person or one government does not seem logical for many reasons. the community of ancient NFTs holders must be racially diverse with a wide geographical spread as this is the legacy that must be properly passed down to future generations.

Overstones is very obsessive in choosing cryptocurrencies. So far, many authentic and valuable currencies have been named (150 Item) and minted. We intend to reserve some slots for emerging currencies that will prove their worth in the future.Of course , they must have effective functionality in blockchain development , security , a strong community of investors and fans and other related matters.This policy will greatly contribute to the attractiveness of the collection and the credibility of its society

SMART CONTRACT
BeaconProxy.sol

```
// SPDX-License-Identifier: MIT

pragma solidity >=0.6.0 <0.8.0;

import "./Proxy.sol";
import "../utils/Address.sol";
import "./IBeacon.sol";

/
* @dev This contract implements a proxy that gets the implementation address for each call from a {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 {
    /
     * @dev The storage slot of the UpgradeableBeacon contract which defines the implementation for this proxy.
     * This is bytes32(uint256(keccak256('eip1967.proxy.beacon')) - 1)) and is validated in the constructor.
     */
    bytes32 private constant _BEACON_SLOT = 0xa3f0ad74e5423aebfd80d3ef4346578335a9a72aeaee59ff6cb3582b35133d50;

    /
     * @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 initializating 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) public payable {
        assert(_BEACON_SLOT == bytes32(uint256(keccak256("eip1967.proxy.beacon"))
- 1));
        _setBeacon(beacon, data);
    }

    /
     * @dev Returns the current beacon address.
     */
    function _beacon() internal view virtual returns (address beacon) {
        bytes32 slot = _BEACON_SLOT;
        // solhint-disable-next-line no-inline-assembly
        assembly {
            beacon := sload(slot)
        }
    }

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

    /
     * @dev Changes the proxy to use a new beacon.
     *
     * 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 {
        require(
            Address.isContract(beacon),
            "BeaconProxy: beacon is not a contract"
        );
        require(
            Address.isContract(IBeacon(beacon).implementation()),
            "BeaconProxy: beacon implementation is not a contract"
        );
        bytes32 slot = _BEACON_SLOT;

        // solhint-disable-next-line no-inline-assembly
        assembly {
            sstore(slot, beacon)
        }

        if (data.length > 0) {
            Address.functionDelegateCall(_implementation(), data, "BeaconProxy: function
call failed");
        }
    }
}
```

Address.sol

```
// SPDX-License-Identifier: MIT

pragma solidity >=0.6.2 <0.8.0;

/
* @dev Collection of functions related to the address type
*/
library Address {
    /
     * @dev Returns true if
account
is a contract.
     *
     * [IMPORTANT]
     * ====
     * It is unsafe to assume that an address for which this function returns
     * false is an externally-owned account (EOA) and not a contract.
     *
     * Among others,
isContract
will return false for the following
     * types of addresses:
     *
     *  - an externally-owned account
     *  - a contract in construction
     *  - an address where a contract will be created
     *  - an address where a contract lived, but was destroyed
     * ====
     */
    function isContract(address account) internal view returns (bool) {
        // This method relies on extcodesize, which returns 0 for contracts in
        // construction, since the code is only stored at the end of the
        // constructor execution.

        uint256 size;
        // solhint-disable-next-line no-inline-assembly
        assembly { size := extcodesize(account) }
        return size > 0;
    }

    /
     * @dev Replacement for Solidity's `transfer`: sends `amount` wei to
     * `recipient`, forwarding all available gas and reverting on errors.
```

```
    *
    * https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas cost
    * of certain opcodes, possibly making contracts go over the 2300 gas limit
    * imposed by `transfer`, making them unable to receive funds via
    * `transfer`. {sendValue} removes this limitation.
    *
    * https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-now/[Learn more].
    *
    * IMPORTANT: because control is transferred to `recipient`, care must be
    * taken to not create reentrancy vulnerabilities. Consider using
    * {ReentrancyGuard} or the
    * https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#use-the-checks-effects-interactions-pattern[checks-effects-interactions pattern].
    */
   function sendValue(address payable recipient, uint256 amount) internal {
       require(address(this).balance >= amount, "Address: insufficient balance");

       // solhint-disable-next-line avoid-low-level-calls, avoid-call-value
       (bool success, ) = recipient.call{ value: amount }("");
       require(success, "Address: unable to send value, recipient may have reverted");
   }

   /
    * @dev Performs a Solidity function call using a low level
`call`. A
    * plain`call` is an unsafe replacement for a function call: use this
    * function instead.
    *
    * If `target` reverts with a revert reason, it is bubbled up by this
    * function (like regular Solidity function calls).
    *
    * Returns the raw returned data. To convert to the expected return value,
    * use https://solidity.readthedocs.io/en/latest/units-and-global-variables.html?highlight=abi.decode#abi-encoding-and-decoding-functions[`abi.decode`].
    *
    * Requirements:
    *
    * - `target` must be a contract.
    * - calling
```

target
with
data
must not revert.
     *
     * _Available since v3.1._
     */
    function functionCall(address target, bytes memory data) internal returns (bytes memory) {
      return functionCall(target, data, "Address: low-level call failed");
    }

/
     ** @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`], but with
     ** `errorMessage` as a fallback revert reason when `target` reverts.
     **
     ** _Available since v3.1._
     **/
    function functionCall(address target, bytes memory data, string memory errorMessage) internal returns (bytes memory) {
        return functionCallWithValue(target, data, 0, errorMessage);
    }

    /
     * @dev Same as {xref-Address-functionCall-address-bytes-}[
functionCall
],
     * but also transferring
value
wei to
target
.
     *
     * Requirements:
     *
     * - the calling contract must have an ETH balance of at least
value
.
     * - the called Solidity function must be
payable
.
     *
     * _Available since v3.1._
     */
    function functionCallWithValue(address target, bytes memory data, uint256 value) internal returns (bytes memory) {
        return functionCallWithValue(target, data, value, "Address: low-level call with value

```
failed");
    }

    /
     * @dev Same as {xref-Address-functionCallWithValue-address-bytes-uint256-}[`functionCallWithValue`], but
     * with `errorMessage` as a fallback revert reason when `target` reverts.
     *
     * _Available since v3.1._
     */
    function functionCallWithValue(address target, bytes memory data, uint256 value, string memory errorMessage) internal returns (bytes memory) {
        require(address(this).balance >= value, "Address: insufficient balance for call");
        require(isContract(target), "Address: call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = target.call{ value: value }(data);
        return _verifyCallResult(success, returndata, errorMessage);
    }

    /
     * @dev Same as {xref-Address-functionCall-address-bytes-}[functionCall],
     * but performing a static call.
     *
     * _Available since v3.3._
     */
    function functionStaticCall(address target, bytes memory data) internal view returns (bytes memory) {
        return functionStaticCall(target, data, "Address: low-level static call failed");
    }

    /
     * @dev Same as {xref-Address-functionCall-address-bytes-string-}[`functionCall`],
     * but performing a static call.
     *
     * _Available since v3.3._
     */
    function functionStaticCall(address target, bytes memory data, string memory errorMessage) internal view returns (bytes memory) {
        require(isContract(target), "Address: static call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = target.staticcall(data);
```

```
        return _verifyCallResult(success, returndata, errorMessage);
    }

    /
     * @dev Same as {xref-Address-functionCall-address-bytes-}[
functionCall
],
     * but performing a delegate call.
     *
     * _Available since v3.4._
     */
    function functionDelegateCall(address target, bytes memory data) internal returns
(bytes memory) {
        return functionDelegateCall(target, data, "Address: low-level delegate call failed");
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-string-}[
functionCall
],
     * but performing a delegate call.
     *
     * _Available since v3.4._
     */
    function functionDelegateCall(address target, bytes memory data, string memory
errorMessage) internal returns (bytes memory) {
        require(isContract(target), "Address: delegate call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = target.delegatecall(data);
        return _verifyCallResult(success, returndata, errorMessage);
    }

function _verifyCallResult(bool success, bytes memory returndata, string memory
errorMessage) private pure returns(bytes memory) {
        if (success) {
            return returndata;
        } else {
            // Look for revert reason and bubble it up if present
            if (returndata.length > 0) {
                // The easiest way to bubble the revert reason is using memory via assembly

                // solhint-disable-next-line no-inline-assembly
                assembly {
                    let returndata_size := mload(returndata)
                    revert(add(32, returndata), returndata_size)
```

```
            }
        } else {
            revert(errorMessage);
        }
    }
  }
}
```

Proxy.sol

```
// SPDX-License-Identifier: MIT

pragma solidity >=0.6.0 <0.8.0;

/
* @dev This abstract contract provides a fallback function that delegates all calls to another contract using the EVM
* instruction `delegatecall`. We refer to the second contract as the _implementation_ behind the proxy, and it has to
* be specified by overriding the virtual {_implementation} function.
*
* Additionally, delegation to the implementation can be triggered manually through the {_fallback} function, or to a
* different contract through the {_delegate} function.
*
* The success and return data of the delegated call will be returned back to the caller of the proxy.
*/
abstract contract Proxy {
    /
     * @dev Delegates the current call to `implementation`.
     *
     * This function does not return to its internall call site, it will return directly to the external caller.
     */
    function _delegate(address implementation) internal virtual {
        // solhint-disable-next-line no-inline-assembly
        assembly {
            // Copy msg.data. We take full control of memory in this inline assembly
            // block because it will not return to Solidity code. We overwrite the
            // Solidity scratch pad at memory position 0.
            calldatacopy(0, 0, calldatasize())

            // Call the implementation.
            // out and outsize are 0 because we don't know the size yet.
            let result := delegatecall(gas(), implementation, 0, calldatasize(), 0, 0)

            // Copy the returned data.
            returndatacopy(0, 0, returndatasize())

            switch result
```

```
            // delegatecall returns 0 on error.
            case 0 { revert(0, returndatasize()) }
            default { return(0, returndatasize()) }
        }
    }

    /
     * @dev This is a virtual function that should be overriden so it returns the address to which the fallback function
     * and {_fallback} should delegate.
     */
    function _implementation() internal view virtual returns (address);

    /
     * @dev Delegates the current call to the address returned by `_implementation()`.
     *
     * This function does not return to its internall call site, it will return directly to the external caller.
     */
    function _fallback() internal virtual {
        _beforeFallback();
        _delegate(_implementation());
    }

    /
     * @dev Fallback function that delegates calls to the address returned by `_implementation()`. Will run if no other
     * function in the contract matches the call data.
     */
    fallback () external payable virtual {
        _fallback();
    }

    /
     * @dev Fallback function that delegates calls to the address returned by `_implementation()`. Will run if call data
     * is empty.
     */
    receive () external payable virtual {
        _fallback();
    }

    /**
```

```
    * @dev Hook that is called before falling back to the implementation. Can happen as part of a manual `_fallback`
    * call, or as part of the Solidity `fallback` or `receive` functions.
    *
    * If overriden should call `super._beforeFallback()`.
    */
   function _beforeFallback() internal virtual {
   }
}
```

IBeacon.sol

```
// SPDX-License-Identifier: MIT

pragma solidity >=0.6.0 <0.8.0;

/
* @dev This is the interface that {BeaconProxy} expects of its beacon.
*/
interface IBeacon {
    /
     * @dev Must return an address that can be used as a delegate call target.
     *
     * {BeaconProxy} will check that this address is a contract.
     */
    function implementation() external view returns (address);
}
```

Settings

```
{
  "remappings": [],
  "optimizer": {
    "enabled": true,
    "runs": 200
  },
  "evmVersion": "istanbul",
  "libraries": {},
  "outputSelection": {
    "*": {
      "*": [
        "evm.bytecode",
        "evm.deployedBytecode",
        "devdoc",
        "userdoc",
        "metadata",
        "abi"
      ]
    }
  }
}
```

Social Networks

https://www.linkedin.com/in/overstones

https://www.reddit.com/user/overstones

https://github.com/overstones

https://discord.com/over_stones

https://twitter.com/over_stones

https://www.instagram.com/over_stones

https://t.me/over_stones

https://medium.com/@metagallery.nft