Smart contracts PoP Docs

Introduction

PoP Docs was developed for you to manage your personal information in a decentralized way, safely and easily.
The app was created to save user data on the Blockchain and give the user the power to control their own data.
The application is developed with a smart contract architecture, created to save and manage user data and the relationship between users and validators.

- Secure storage of personal information
- Safely validated information
- Authorized validators
- Published owner-approved certificates

Smart Contracts

PoP Docs smart contracts are Ownable and Role-Based Access Control.

The Ownable module provides a basic access control mechanism, where there is an account (an owner) that can be granted exclusive access to specific functions.

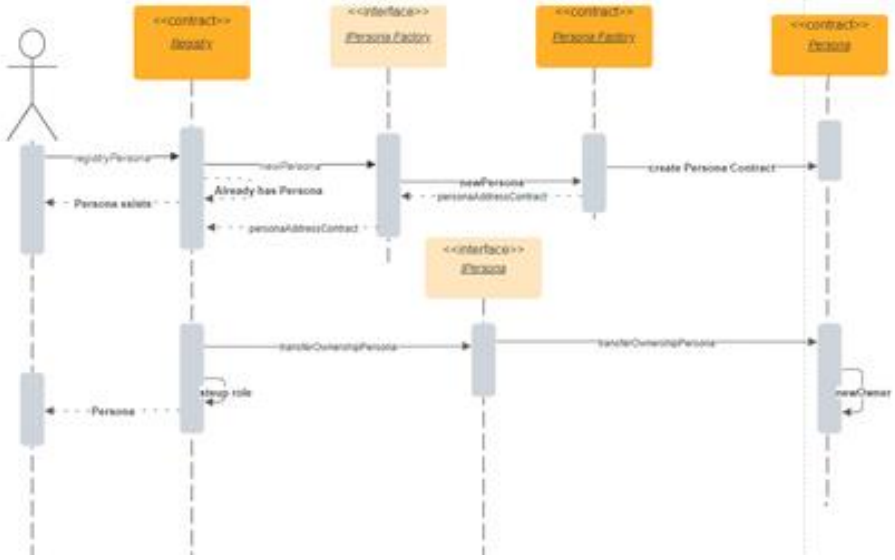
The Access Control module allows you to implement a role-based access control mechanism.

The Registry smart contract is responsible for maintaining the ecosystem of roles. It communicates with the Factories Persona and Validator interfaces to create new records.

All his communication is done through the smart contract interfaces.

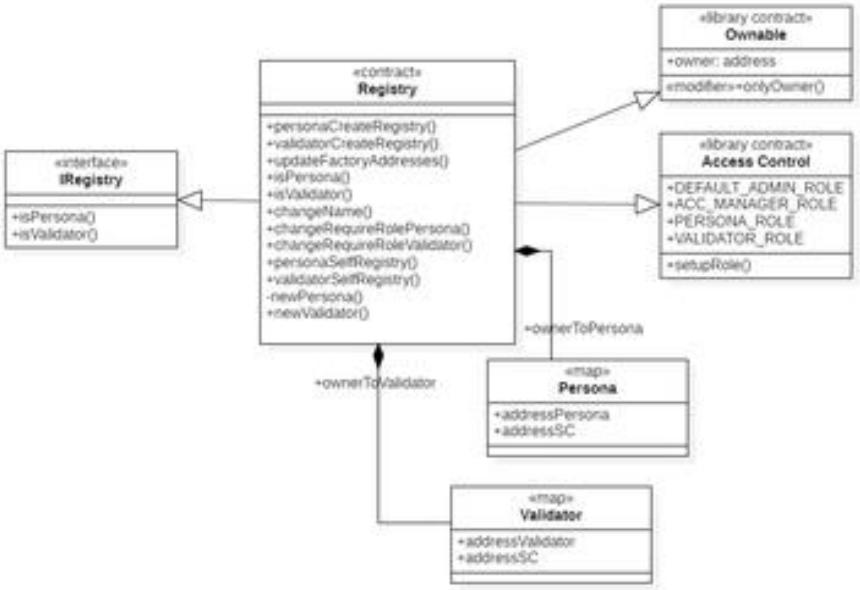
This way it facilitates the update process if necessary.

Below is a demonstration of the communication between smart contracts to create Persona contract:



Registry

The Registry smart contract is responsible for creating Persona and Validator by communicating through the Persona Factory and Validator Factory interfaces.

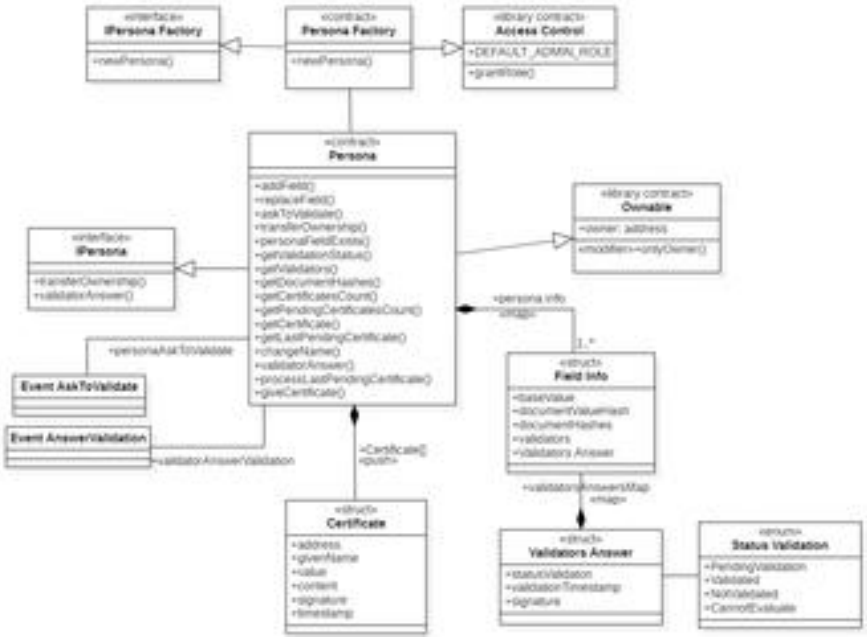


Functions

Function	Parameters	Description
personaCreateRegistry	`address`, `name`	Check if the address is already registered as a Persona, if not, ask Persona Factory to create the Persona smart contract and add the address to the Persona role.
validatorCreateRegistry	`address`, `name`	Check if the address is already registered as a Validator, if not, check with the Validator Factory if the address has permission to create the Validator smart contract, if the address has permission, the Validator smart contract is created and added to the Validator role.

Persona

The Persona smart contract is responsible for adding Persona registries and certificates, managing the access of the authorized validator to Persona registries and thus validating the information.



Functions

Function	Parameters	Description

addField	`onlyOwner`
	
	`documentType, documentValueHash, memory documentHashes`
		The Persona contract owner can add registry in your smart contract.

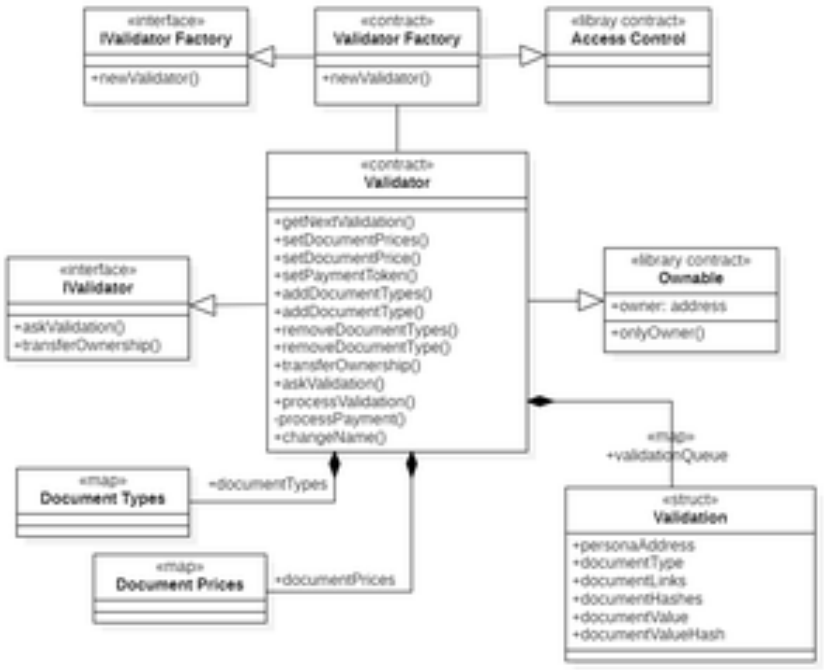
replaceField	`onlyOwner`
	
	`documentType, documentValueHash, memory documentHashes`
		The Persona contract owner can change the validator of your registry in your smart contract.

askToValidate	`onlyOwner`
	
	`validatorAddress, documentType, memory documentLinks, memory documentValue`
		The Persona contract owner can define the validator of information in your smart contract and ask to validate your registry.

validatorAnswer	`documentType, status, timestamp, memory signature`	The authorized validator can use this function available in the Persona interface to select one of the options: Not Evaluated, Validated, Not Validated for Persona registry.
giveCertificate	`givenName, value, content, signature`	Add a new pending Certificate.

Validator

The validator's smart contract is responsible for storing the record queues for the validator to validate and the validator service configuration.



Functions

Function	Parameters	Description

setDocumentPrices	`onlyOwner`
	
	`memory _documentTypes, memory _documentPrices`
		Configures the amount to be paid per document type.

setPaymentToken	`onlyOwner`
	
	`address token`
		Configures the currency to be accepted for payment.

addDocumentTypes	`onlyOwner`
	
	`memory _documentTypes`
		Adds types of documents that the validator is able to validate.

removeDocumentTypes	`onlyOwner`
	
	`memory _documentTypes`
		Removes documents from the list of documents suitable for validation.

askValidation	`onlyOwner`
	
	`documentType`, `documentLinks`, `documentHashes`, `documentValue`, `documentValueHash`
		Adds a record to the validator's pending queue to be validated.

processValidation	`status`, `memory signature`	The validator accesses the queue and responds to validation