# Auctus Technical Paper

Version 1.0 - 6 February 2018

Felipe Silveira, Ariny Guedes, Thiago Araújo, Daniel Figueiredo

## 1 – Platform Architecture Overview

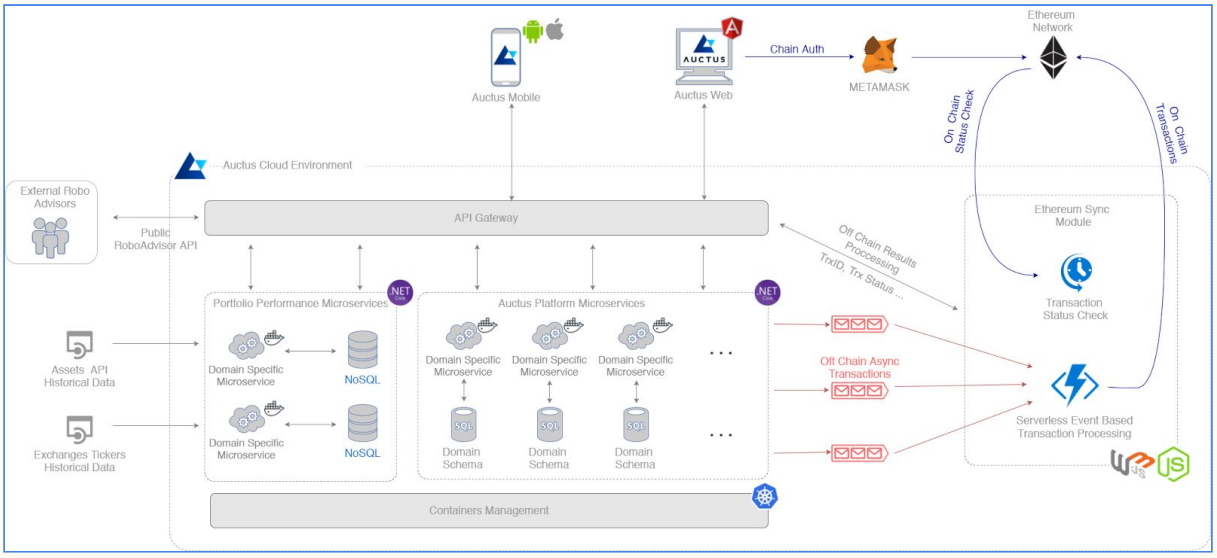
Our main architectural goal at Auctus is to build a platform which is able to sustain high traffic of a global user base. Therefore, the platform should be able to proper escalate on heavy load moments and, thus, guarantee a fluid and robust experience for our users or API consumers.

Auctus development team, based on previous experiences of similar platforms and market best practices, carefully evaluated the best development practices combining cloud / containerized computing, micro-services and the emerging Ethereum Network technologies.

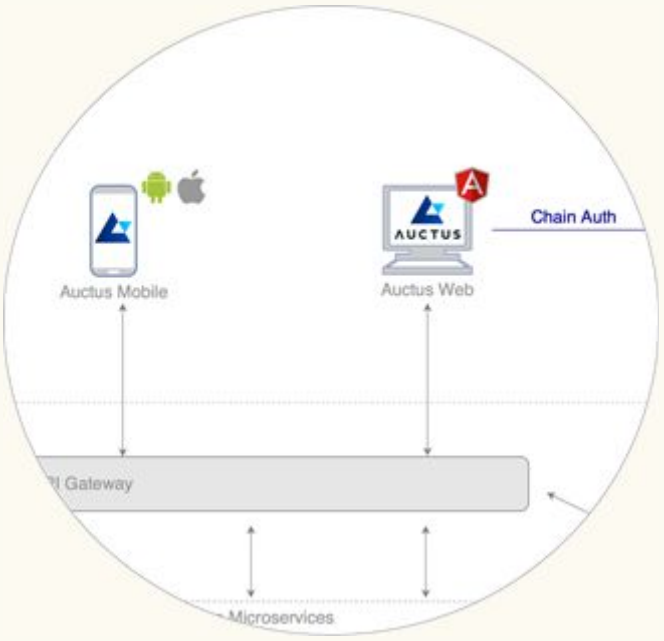
### Cloud Specific Implementation vs Containerized Microservices

Despite the fact that for strategic purposes, we have chosen to host our platform at Microsoft Azure, we believe that it is a good practice to be technology-agnostic[1]. This approach will guarantee that our entire platform can be fully migrated to a new cloud provider, or every single module / service can have a painless and seamless technology update if needed.

The overview of this architecture is depicted in the diagram below:




[1] Newman, S. (2015). Chapter 4: Integration In *Building microservices* (pp 39-78). Sebastopol (CA): OReillyMedia




## Auctus Platform FrontEnd

The Auctus Platform front end will be built for both web and mobile devices.

The web platform will be built using Angular 5.

At first, the mobile platform will be built natively for iOS and Android powered devices, however, for development time reduction, we will create some POCs in order to evaluate cross-platform solutions like Xamarin, Ionic or React Native.

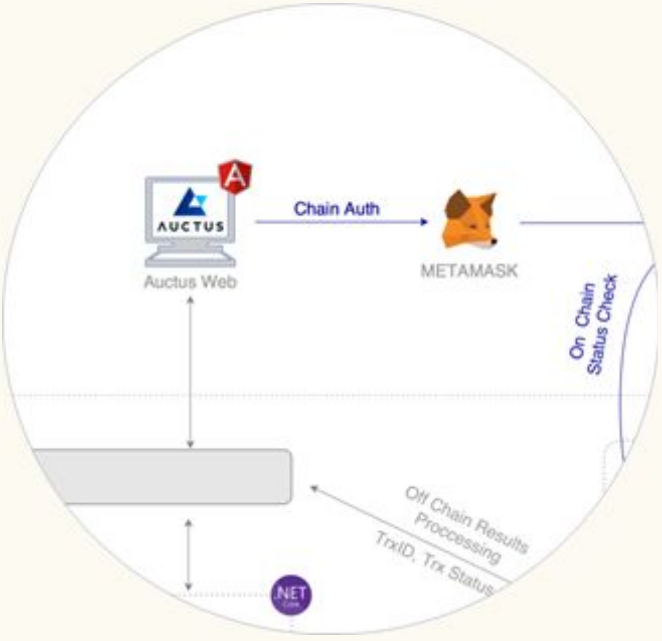
Both web and mobile clients will be built as a thin client and will have access to Auctus business layer through a API Gateway which will be described in more details next.

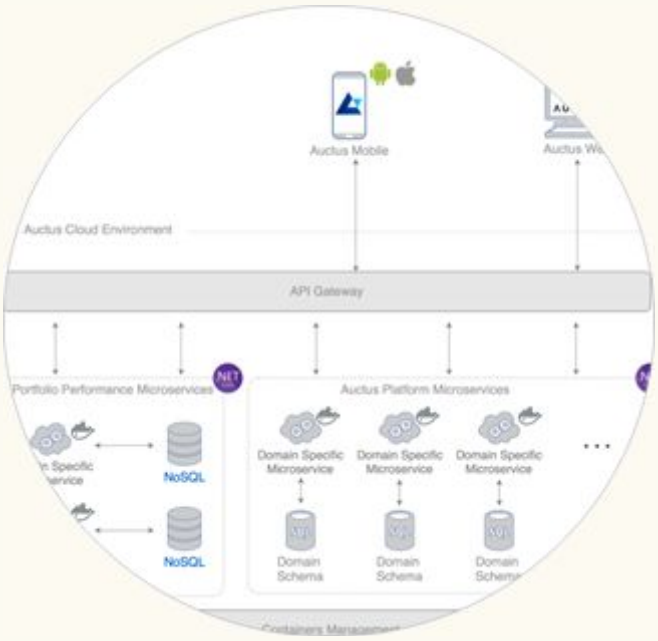
## Authentication & Authorization

The user authentication will be performed in two ways:

OAuth2 SSO for platform off-chain access and METAMASK for accessing Ethereum Network integration features.

The Auctus platform will provide different usage plans for its users, and the authorization (the ability of users to access different features of the platform) will be a combination of a profile schema and AUC token escrow.






## API Gateway

The API Gateway will provide an integration abstraction layer which will expose, through RESTful APIs, all the services needed to interact with the platform backend.

The API Gateway layer will be an important tool to manage our set of services, and will also provide a set of features which will help us to secure, control and scale our APIs.

These management features include:

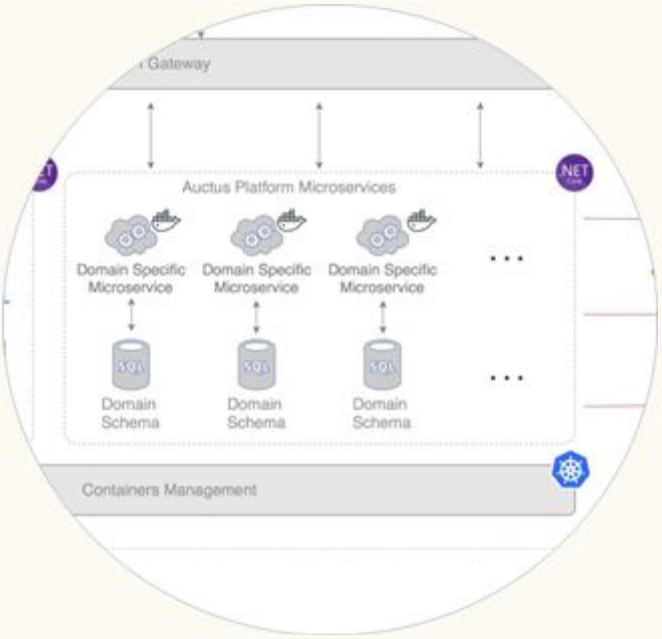
- Service Authentication
    - Private API for Auctus FrontEnd
    - Public API for Robo Advisor API
- Response Caching
- Service Transformation
- Request Logging, Usage, Performance and Health Analytics
- API Versioning

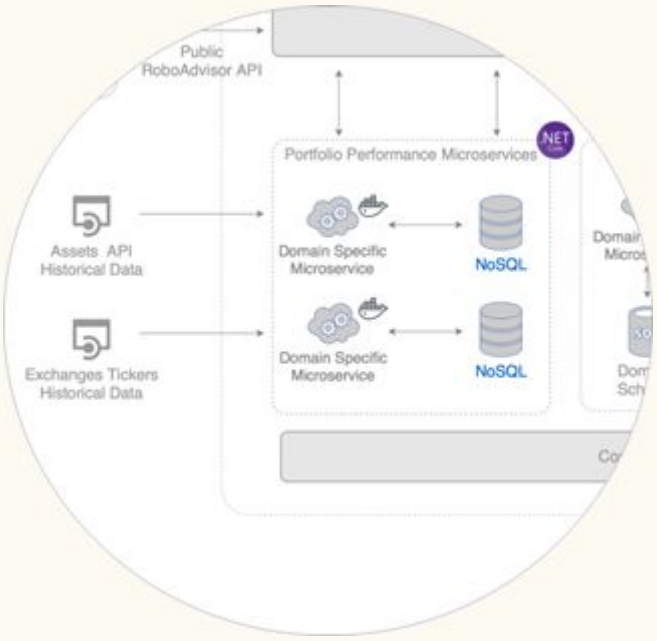
## Auctus Platform Microservices

The backend module of Auctus platform will consist of a set of microservices developed using the ASP.NET Core framework and hosted on docker containers.

This module provides the services to support the mobile app and web application. It will be built under the premises of DDD (domain driven design) in order to provide decoupling and domain data ownership.

It will contain the rules for server operations related to the platform, such as portfolio creation and performance tracking.






## Portfolio Performance Microservices

To track historical performance of recommended portfolios, it is necessary to get information from outside the platform. The portfolio results module will get market information using third-party APIs like:

- Exchange API for cryptocurrency historical values
- Market history API for traditional assets.

Due to the schemaless characteristic of the retrieved data, summed with the potential size of some years of historical information and the need of normalization, the data collected in this module will be stored on a Non-Relational Database (NoSQL).

This data is intended to be used for historical charts in the platform.

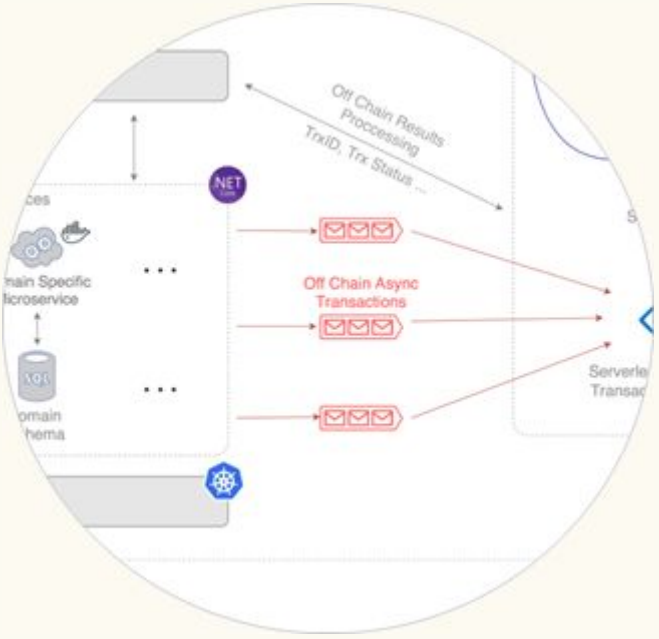
At first, we will be using the document-based database MongoDB, which will provide us JSON (BSON) normalization of the retrieved data, facilitating the integration with chart components.

## Off Chain Async Transactions

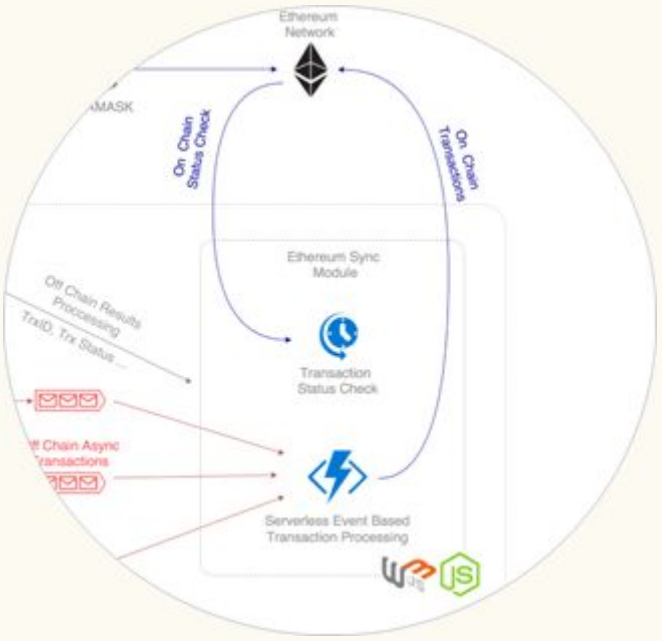
This layer will be responsible for providing decoupling between the microservices layer and the Ethereum network integration.

The main objective of this layer is to improve the Auctus' Platform user experience and guarantee transaction deliver.

Through asynchronous transaction integration to the blockchain, we will prevent locks, (which could cause response delays) and, therefore, improve the response time of the application.




This approach will make the solution more scalable and resilient to eventual Ethereum network instability or heavy load.




## Ethereum Sync Module (On-Chain Transactions)

This module, which will be built using NodeJS and web3.js for Ethereum Network integration, will consist of two parts:

**Serverless Event-Based Transaction Processing**:
Responsible for receiving a message from the queue and sending the transaction to the Ethereum node. After posting the transaction and retrieving its ID and metadata, this processor will invoke an API to consist that information into Auctus database. In order to guarantee the optimum usage of our infrastructure, we will be using serverless event processing (e.g. Azure Function, AWS Lambda).

**Transaction Status Check:**
Responsible for pooling the Ethereum node checking the pending transaction status until they are completed. After completion, this processor will invoke an API to update the transaction status.

**Advice Match Mechanics:**
Some of the application information will be stored in smart contracts, developed using Solidity.

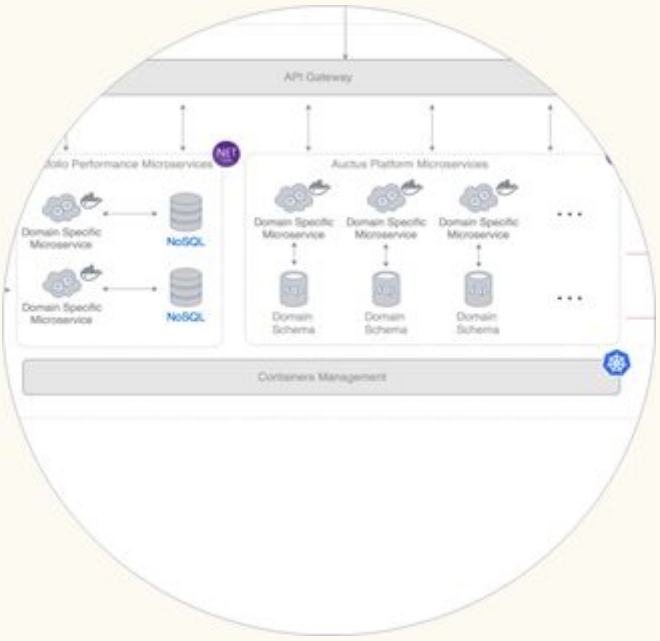
Payments for advice will be made using smart contracts, as well as the portfolio projection values.

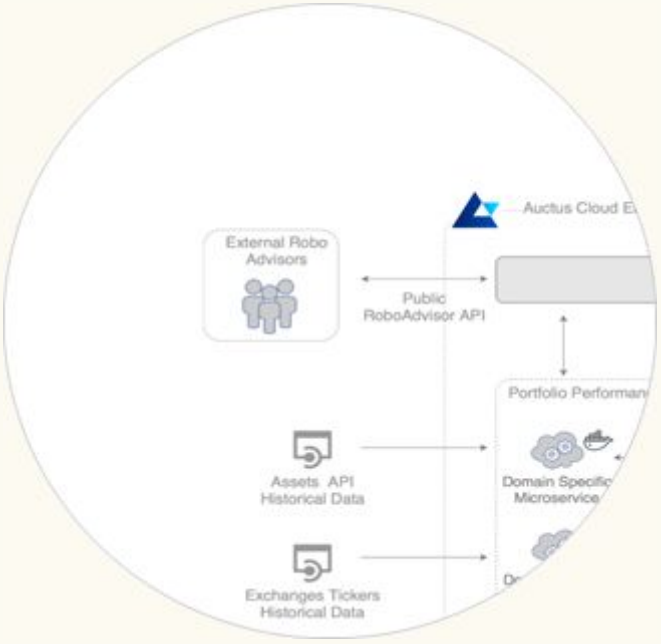
A smart contract-based oracle service, such as Oraclize (http://www.oraclize.it/) will be used to collect the real portfolio performance after a defined period. This information will define if the tokens locked for advice payment will either be transferred to the advisor (in case the advice was good, meaning that results met predictions) or redeemed by the customer (in case of bad advice).

## Container Management and Auto Scaling

This layer will be responsible for the DevOps of Auctus' microservices environment.

We will use Kubernetes to manage the Docker containers, provide auto-scaling, configuration management and automated deployment.




## Robo Advisor Public API

The API gateway layer will provide a public API, that will enable Auctus' Platform users and third party financial specialists to connect Robo advisors, which will be available at our marketplace.

# 2 – Smart contracts

The backend of the Auctus Platform is built upon the Ethereum Blockchain using smart contracts developed using the Solidity programming language as the main mechanism to provide automation, transparency and a result-oriented fee structure.

The three main smart contracts that compose the solution are the following:

## The AUC token smart contract

The smart contract that defines the AUC token is compliant with the ERC-20 and ERC-223 standards.

```
pragma solidity ^0.4.19;

contract ERC20 {
    uint256 public totalSupply;
    uint8 public decimals;

    function balanceOf(address who) public constant returns (uint256);
    function allowance(address owner, address spender) public constant returns (uint256);
    function transfer(address to, uint256 value) public returns (bool);
    function approve(address spender, uint256 value) public returns (bool);
    function transferFrom(address from, address to, uint256 value) public returns (bool);

    event Transfer(address indexed from, address indexed to, uint256 value);
    event Approval(address indexed owner, address indexed spender, uint256 value);
}


contract ERC223 {
  uint256 public totalSupply;
  uint8 public decimals;
  string public name;
  string public symbol;

  function balanceOf(address who) public constant returns (uint256);
  function transfer(address to, uint256 value) public returns (bool);
  function transfer(address to, uint256 value, bytes data) public returns (bool);
  function transfer(address to, uint256 value, bytes data, string custom_fallback) public returns (bool);

  event Transfer(address indexed from, address indexed to, uint256 value, bytes indexed data);
}
```

```
contract AuctusToken is ERC20, ERC223 {
    uint256 public totalSupply;
    uint8 public decimals = 18;
    string public name = "Auctus Token";
    string public symbol = "AUC";
```

## The AuctusEscrow main smart contract

The AuctusEscrow smart contract is used for locking or redeeming AUC tokens. This feature is used for accessing the platform.

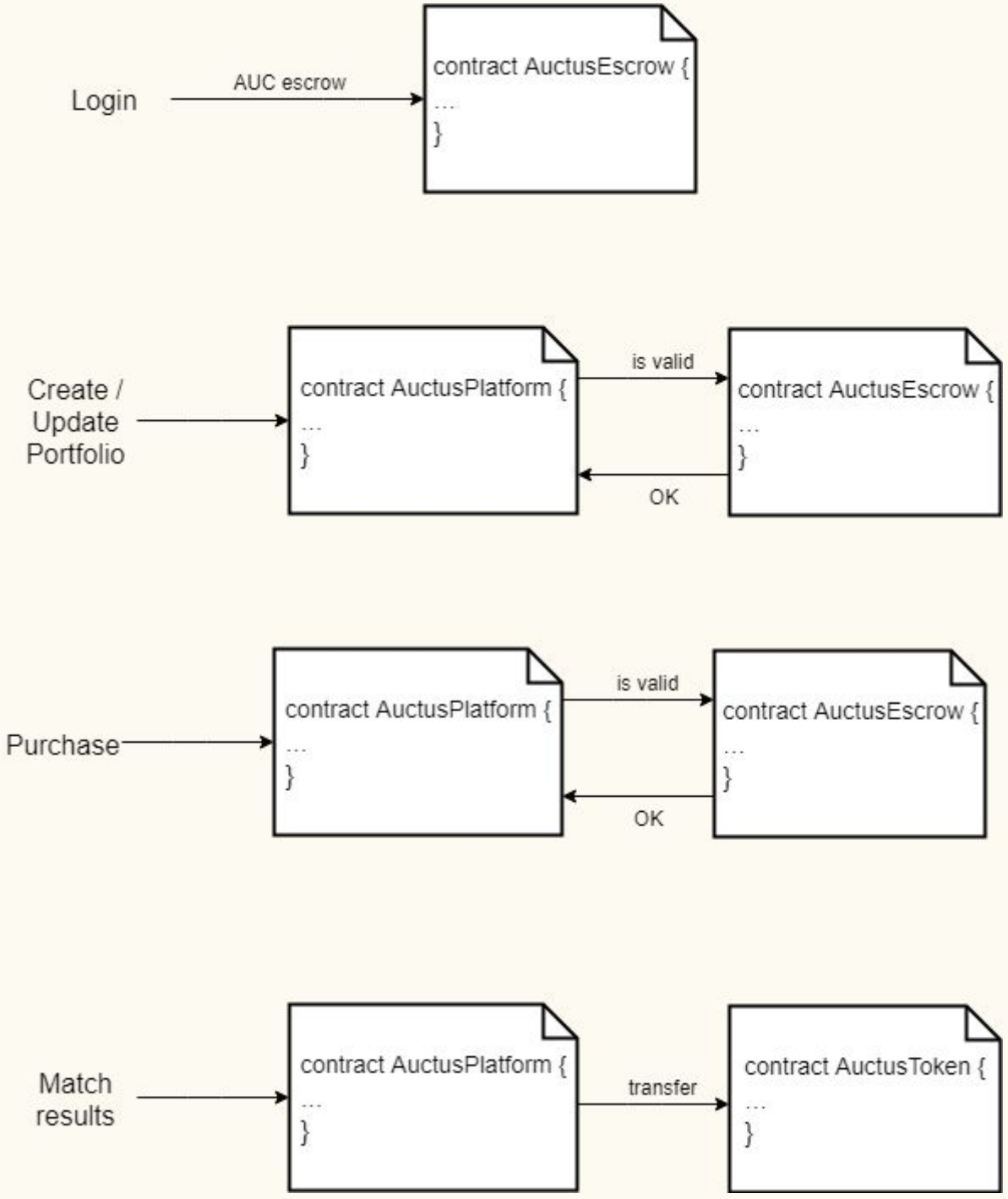
```
pragma solidity ^0.4.19;

contract ContractReceiver {
  function tokenFallback(address from, uint256 value, bytes data) public;
}

contract AuctusToken {
  function transfer(address to, uint256 value) public returns (bool);
}

contract AuctusAlphaEscrow is ContractReceiver {
    using SafeMath for uint256;

    event Escrow(address indexed from, uint256 value);
    event Redeem(address indexed from, uint256 value);
```

## The AuctusPlatform smart contract

The AuctusPlatform smart contract interacts with the token and escrow contracts. It is the main smart contract of the platform, used for recording the projections made by financial advisors. It orchestrates the purchase of products and services using the AUC tokens in the decentralized marketplace taking into consideration other business rules such as the matching of predictions with real results.

```
pragma solidity ^0.4.19;

contract AuctusToken {
  function transfer(address to, uint256 value) public returns (bool);
}

contract AuctusEscrow {
  function isValidEscrow(address who) public constant returns (bool);
}

contract ContractReceiver {
  function tokenFallback(address from, uint256 value, bytes data) public;
}

contract AuctusPlatform is ContractReceiver {
    using SafeMath for uint256;

    struct Portfolio {
        address owner;
        uint256 projection;
        uint256 price;
        uint64 period;
        bool enabled;
        string distribution;
    }

    struct Purchase {
        uint256 identifier;
        uint256 price;
        uint64 period;
```

**Use cases illustration**



# 3 - Technology-specific disclaimer

Technology is changing rapidly. The description above is based on Auctus current development and on the current state of technology. The described architecture and solution will be implemented in an incremental manner and might not be fully developed in the first releases. The presented examples of code are purely for illustration purposes. Auctus development team reserves the right to change approaches, architecture and use of smart contracts in case the team finds more suitable options to solve the technical needs of the platform in future releases.