Table of Contents

Thesis

Appendix

Melon Reporting and Auditing

tbd

Bachelorthesis by Simon Emanuel Schmid and Benjamin Zumbrunn

Spring Semester 2018

University of Applied Sciences Northwestern Switzerland FHNW, School of Engineering. BSc Computer Sciences / iCompetence.

Coaches: Prof. Dr. Sarah Hauser, Markus Knecht

Customer: Melonport AG

Brugg, August 16th 2018

TODO: Logo Melonport & FHNW

Abstract

TODO: Describe what we achieved (not how!)

Introduction

Organization

The main entry point for this project is the open source Github repository github.com/melonproject/reporting-thesis.

The documentation can be found (and commented) on Gitbooks: schmidsi.gitbooks.io/melon-reporting.

Project management is done with a simple Kanban board as a Github Project: github.com/melonproject/reporting-thesis/projects/1

Single tasks are managed as Github Issues: github.com/melonproject/reporting-thesis/issues

/

Problem statement

Vision

Creating functionality on top of the Melon protocol that automates reporting/auditing almost completely:

> a) Something that a real fund manager would be able to confidently say: “This solves my reporting issues and makes my life a lot easier”
>
> b) Something that can be show-cased to FINMA (and other regulators) and show them how: "This will make *their* life over-seeing a lot easier”

Hypothesis

It is possible to extract and visualize all relevant data from the Melon protocol on the Ethereum blockchain in a way that could be legally acceptable by regulators. Furthermore, this data can be audited and digitally signed and a track record of these audits can be placed on the blockchain again.

Boundaries

Legal

This is a technical thesis and therefore we do not deeply research into the legal aspects of fund management and reporting. But we will find ways how technology can support the legal processes.

Technical

A lot of functionality of funds depends on third party modules: Price feeds, participation, exchanges. In this thesis, we only guarantee the official modules provided by Melonport. If modules outside of this scope could provide important functionality we document this, but do not implement the necessary bridges.

We try to adapt to the latest Melon smart contracts but fall back to the last known working ones if the latest causing too much problem. Last knonw working version is v0.7.0.

Method

Document organization

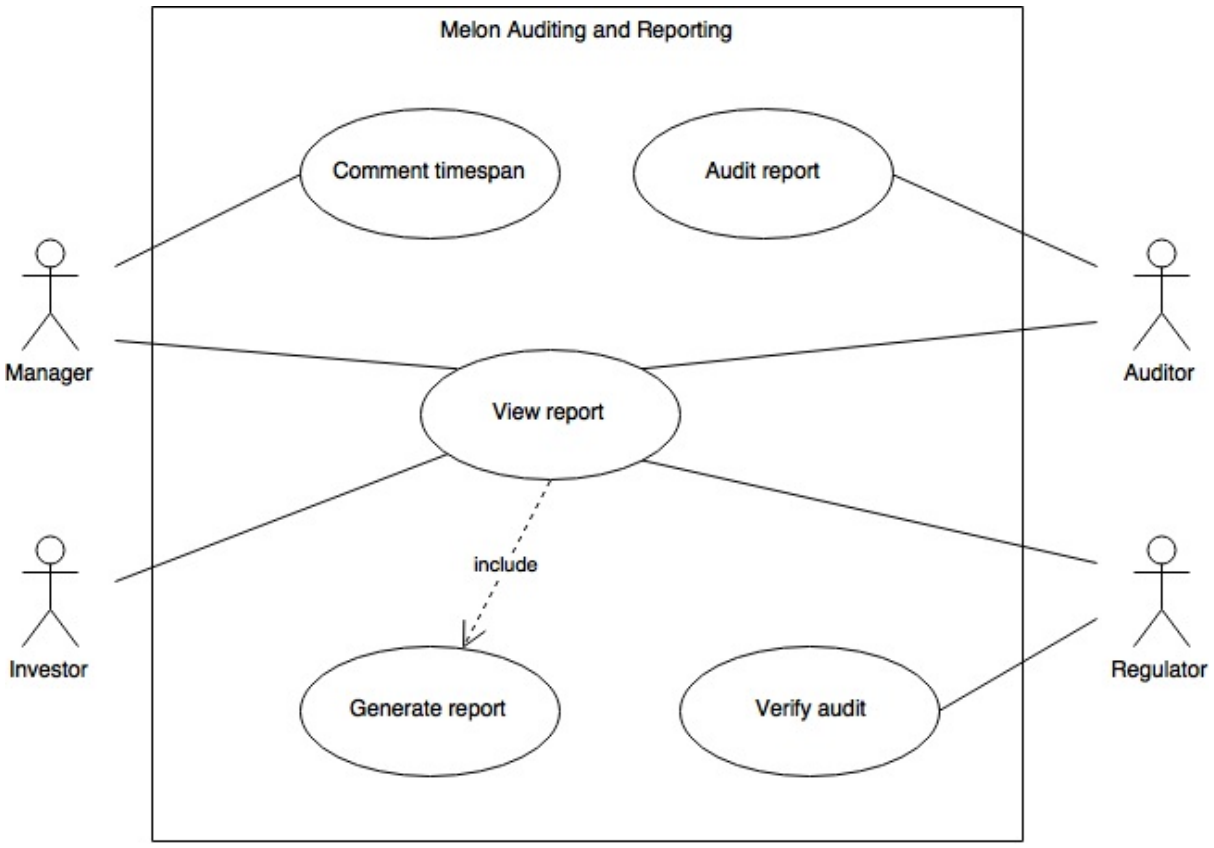
Background

Current State of Fund Reporting

Analysis

Use Cases

Use case diagram



Use case description

Actors

Auditor

Manager

Investor

Regulator

View report

All actors (Auditor, Manager, Investor, Regulator) can view reports about a melon fund. The actor defines a timespan to determine what data is in the scope.

A report consists of the following attributes:

Basic Information

- Fund Name
- Start timestamp of data
- End timestamp of data
- Ticker
- NAV
- AUM
- Date of Inception
- Management Fee
- Performance Fee

Performance

- Since Inception
- YTD
- 1 year
- 2 year
- 3 year
- Volatility (1 year)
- Share ratio

Audits

- Signature
- Name of auditor (from address)
- Timespan of audit

Additional data

- Overview (fund description)
- Historical performance
- Monthly performance
- Hash of dataset
- Timeline of past audits (optional)

Generate report

The report is generated "on the fly".

Input arguments are:

- Fund (address, name?)
- Timespan (timestamp to timestamp)

When a report is generated with the same arguments, the datahash of the report will always be the same.

Audit report

When an auditor has reviewed a report, he can create an audit on the blockchain.

Input arguments are:

- Fund address
- Datahash of report

Comment timespan

A fund manager can add a comment to the fund over a specific timespan.

Verify audit

A regulator can verify an audit. This is basically the ability to make a spot check of audits.

Data classification

Classifications

Data Type

- **Short text**: A name, some key words or a character.
- **Long text**: A long text, usually written in prose.
- **Number**: Any representation of a number, e.g. a price or a percentage.
- **Address**: Data that is considered to be an address, e.g. an Ethereum address.
- **Timestamp**: Any format representing time. This is mostly a unix timestamp, so the accuracy is at most one second.

Source

- **On-chain call**: Data can be retrieved from the blockchain directly by calling a function. *Examples*: current price, fund name, recent trades, ...
- **On-chain event**: Data can be retrieved from the blockchain by searching for events. This data cannot be used by smart contracts. I.e. a smart contract cannot check if a certain event happened. *Examples*: Price history
- **Off-chain**: Data that is not retrievable from the blockchain.
- **Free text**: General text. Usually lawyer jargon.
- **Given**: Data that is given by a template/recommendation,

Availability

- **Simple**: Data is directly available through a simple (API-) call. *Example*: Fund name, AUM, asset prices, orderbook ...
- **Complex**: Data is generally available but we need to build special tools to extract it. *Examples*: Historical prices
- **N/A**: Data is not available (yet)

Consistency

- **Static**: Once written this data does not change anymore. *Examples*: Fund name, ...
- **Configuration**: Data that is configured out of a predefined set of possible options.
- **Generated**: Data that is generated automatically. *Examples*: Address, indexes, ...
- **Live**: Data that can change frequently. *Examples*: Prices, orderbook, AUM, ...
- **Archive**: List data that stays once written. Archive data can be consolidated for display purposes but needs to be preserved. *Examples*: Trade history, participation history, ...
- **Historic**: List data that stays once written but it's importance/density lessens with age: *Examples*: Price history; It is important to know the exact price for every minute for the last 24h but older prices can be stored in less density, e.g every hour. The older the data, the lesser the density.

Red lines: saved data, black lines: available data

Data

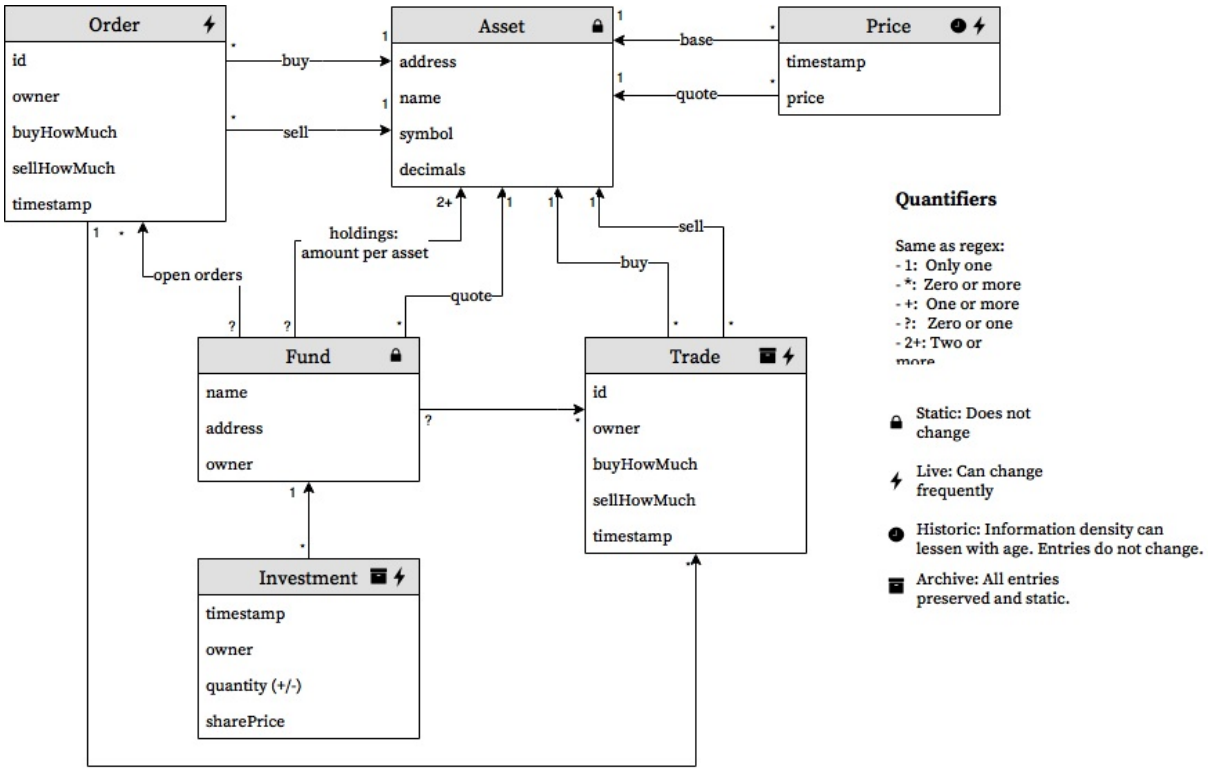
Data	Data Type	Source	Availability	Consistency
Fund name	Short text	On-chain call	Simple	Static

Reasoning

The reasoning behind this classification is described in Appendix D.

On-Chain data

The following ERM is a representation of the minimalistic data available on the blockchain of a fund to build charts and reports. All data can be derived from this model. It is still a challenge to obtain historical data (price feed history) since we need to search the blockchain for events. But this is subject to another chapter. For research purposes it is sufficient to know that this data is available.



Explanation

- The fund is in the center. Once set up, a fund cannot change its name/address/owner
- Every fund has one quote asset in which its value is denominated. Usually MLN.
- Total number of shares (s) can be retrieved with the following formula:

$$s = \sum_{i \in I} q_i$$

  Where I is the set of all investments and q_i the quantity of shares per investment.

- AUM (aum) can be retrieved with the following formula:

$$aum = \sum_{a \in A} h_a * p_a$$

  Where:

  - A is the set of all assets a fund is invested in
  - h_a the amount of holdings the fund has of asset a
  - p_a the price of asset a

- Shareprice (sp):

$$sp = \frac{aum}{s}$$

Off-Chain data

Solution

System Overview

Software Architecture

https://lernajs.io

Screen/Paper Design

Interchangeable Fund Data Format

```
{
  "name": "Example Fund",
  "inception": "yyyy-mm-dd hh:mm:ss",
  "description": "This fund is high risk",
  "manager": "0xbad...a55",
  "nav": 1000,
  "quoteSymbol": "MLN",
  "gav": 1100,
  "timestamp": "yyyy-mm-dd hh:mm:ss",
  "holdings": [
    {
      "symbol": "ETH",
      "amount": 1000
    }
  ],
  "trades": [
    {
      "buySymbol": "ETH",
      "sellSymbol": "MLN",
      "buyAmount": 100,
      "sellAmount": 50,
      "timestamp": "yyyy-mm-dd hh:mm:ss",
      "market": "0xdead...beef"
    }
  ],
  "audits": [
    {
      "timestamp": "yyyy-mm-dd hh:mm:ss",
      "auditor": "0xdead...beef1",
      "dataHash": "QmXZcdco6wZEA2paGeUnoshSB4HJiSTDxagqXerDGop6or",
      "signature": "0x23rasdfasdlfjhasldkfhas"
    }
  ]
}
```

Linked issues:

- https://github.com/melonproject/reporting-thesis/issues/7

Melon Auditing Contract Standard

```
MIP: 1
Title: Melon Auditing
Author: Benjamin Zumbrunn, benzumbrunn@gmail.com
Status: Draft
Type: MIP
Created: 2018-04-24
Reference implementation: https://github.com/melonproject/reporting-thesis/blob/master/packages/contracts/Auditing.sol
```

Abstract

This MIP describes standard functions an auditing contract must implement in order to be compatible as a Melon Auditing module.

Background

The auditing process goes like this:

- A clearly defined algorithm can extract on-chain and off-chain data (I.e. meta data on IPFS) from a fund into a well defined format for a specified timespan.

  `generateData(fundAddress, timespanStart, timespanEnd): FundReportJSON`

  - *Invariant*: A call to `generateData` with the same arguments should always return the same data.
  - *Note*: To generate a report over the whole lifetime of a fund, set: `timespanStart = fund.inception` and `timespanEnd = now`
- An auditor performs an audit on that data and if everything is good he signs the hash of this data on the blockchain.
- Who actually can perform audits on a fund is considered an implementation detail of contracts following this standard.

Motivation

Different fund setups have different auditing requirements:

- Some want to provide a whitelist of auditors to perform and sign audits.
- Others want to implement complex functionalities with identity lookups.
- Some just want to open it for everybody or do not want any audits at all.

In order to follow the modularisation approach of the Melon smart-contract system we hereby define a standard so that minimal requirements can be met. This leads to an open system where module developers can implement their auditing functionality according to their needs outside of our imagination.

Contract details

- The `_auditor` is represented by his/her address.
- The `_dataHash` is the hash of the Interchangeable Fund Data of a report.

- `_timespanStart` and `_timespanEnd` are required to generate the report that was audited.

Transactions

add

```
function add(address _fundAddress, bytes32 _dataHash, uint256 _timespanStart, uint256 _timespanEnd) returns (bo
ol)
```

Add a new audit of a melon fund to the blockchain. **`msg.sender == auditor`** **.**

Calls

isComplete

```
isComplete(address _fundAddress, uint256 _timespanStart, uint256 _timespanEnd) returns (bool)
```

Checks if a fund is completely audited for a given timespan.

Risk management might prevent trades if a fund is not completely audited. For example:

A fund that is audited every month could have the following lookup in risk management: `isComplete(0xfundaddress, fund.inception, now - 30*24*60*60)`

getLength

```
function getLength(address _fundAddress) constant returns (uint256 index)
```

Get the length of the audit array for a specific melon fund. This is needed to iterate through audits with **`getByIndex()`** **.**

getByIndex

```
function getByIndex(address _fundAddress, uint256 _index) constant returns (bytes32 dataHash, address auditor,
uint256 timestamp)
```

Get the stored information of an audit.

Requires that `_index` is smaller than the size of the audit array.

Events

Added

```
event Added(address _fundAddress, uint256 _index)
```

Triggered when a new audit is stored successfully.

Reference Implementation

Auditing.sol

In the reference implementation, we require that only approved auditors can use the method `add()`. The approved auditors are supplied on contract creation via the constructor.

Auditing Contract

function isApprovedAuditor

Auditing.isApprovedAuditor(_auditor) `view` `51c8151a`

Inputs

type	name	description
address	_auditor	undefined

function getByIndex

Auditing.getByIndex(_fundAddress, _index) `view` `834389c7`

Returns the requested audit data

Inputs

type	name	description
address	_fundAddress	undefined
uint256	_index	undefined

function getLastIndex

Auditing.getLastIndex(_fundAddress) `view` `83e714ad`

Returns the last index of a specific fund

Inputs

type	name	description
address	_fundAddress	undefined

function exists

Auditing.exists(_fundAddress, _auditor, _dataHash) `view` `c7c39870`

Validates that the provided data is mapped to an existing audit

Inputs

type	name	description
address	_fundAddress	undefined
address	_auditor	undefined
bytes32	_dataHash	undefined

function add

Auditing.add(_fundAddress, _dataHash, _timespanStart, _timespanEnd) `nonpayable` `ec06b1be`

Creates a new audit on a fund specified with `_fundAddress`, the hashed data in `_dataHash` and the timespan timestamps in `_timespanStart` and `_timespanEnd`.

Inputs

type	name	description
address	_fundAddress	undefined
bytes32	_dataHash	undefined
uint256	_timespanStart	undefined
uint256	_timespanEnd	undefined

function fundAudits

Auditing.fundAudits(,) `view` `f8b79ea9`

Inputs

type	name	description
address		undefined
uint256		undefined

function approvedAuditors

Auditing.approvedAuditors() `view` `f909c6ca`

Inputs

type	name	description
uint256		undefined

event Added

Auditing.Added(_fundAddress, _index) `446e00ad`

Arguments

type	name	description
address	_fundAddress	not indexed
uint256	_index	not indexed

Conclusion

Findings

Future work

Reflection

Technical Quickstart

Project management

Project Goals

Interchangeable Fund Data Format

Auditing Smart Contract Interface Specification

Auditing Smart Contract Interface Reference Implementation

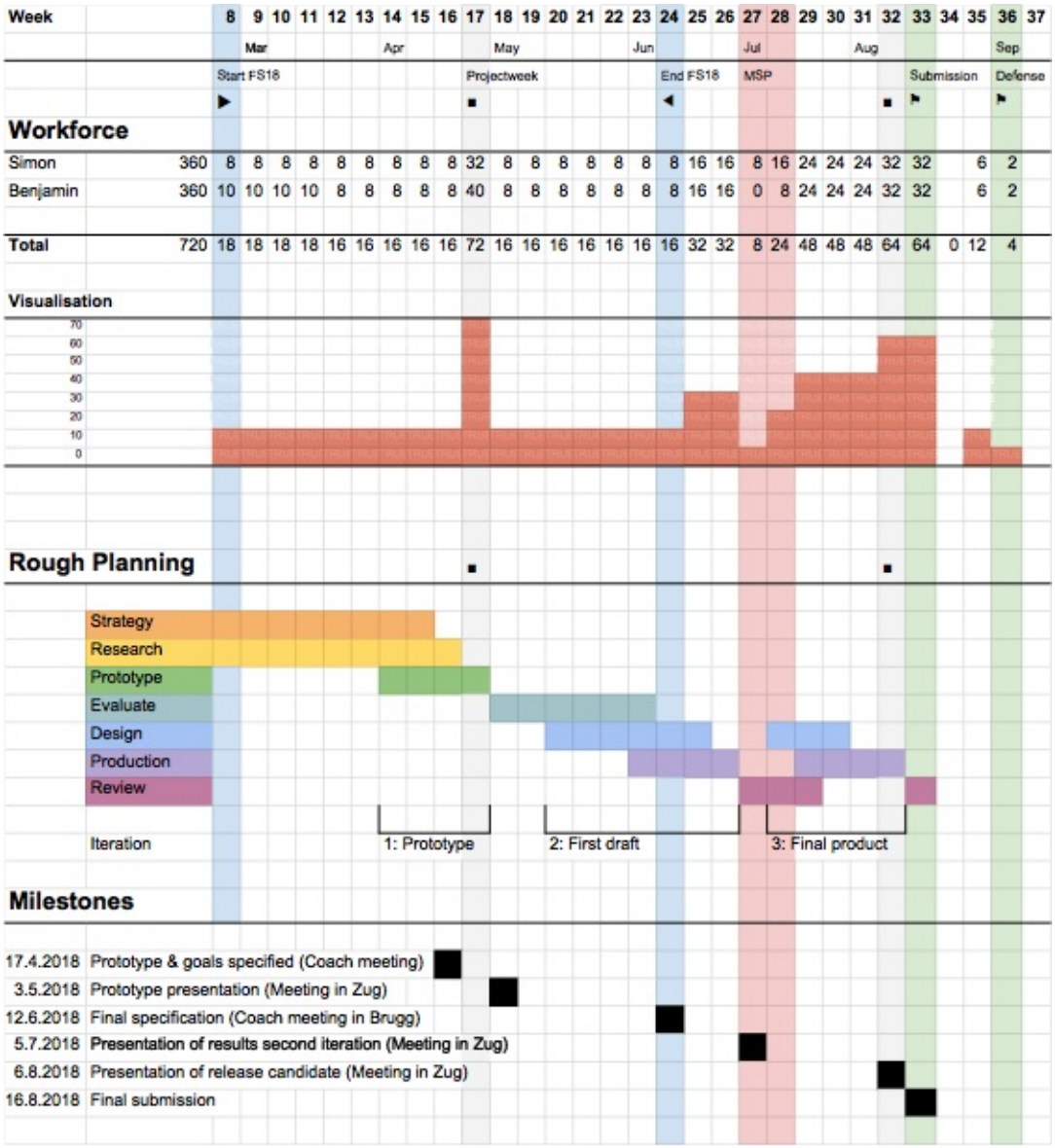
Interchangeable Fund Data Extraction and Consolidation Service

Fund Report Web Interface With Auditing Functionality

Summary of Meetings

- 27.3.2018 - 3pm - In Brugg with coaches: 5.2B31
- 17.4.2018 - 3pm - In Brugg with coaches: 5.2B31
- 3.5.2018 - 2pm - In Zug with all stakeholders except Sarah Hauser
- 22.5.2018 - 4pm - In Brugg with Markus Knecht: 5.2B31
- 12.6.2018 - 3pm - In Brugg with all coaches: 5.2B31
- 5.7.2018 - 2pm - In Zug with all stakeholders
- 17.7.2018 - 3pm - In Brugg with coaches: 5.2B31
- 6.8.2018 - 2pm - In Zug with all stakeholders
- 13.8.2018 - 3pm - In Brugg with coaches: 5.2B34

Planning



Strategy

In a first step the foundation of the project is laid. Vision, hypothesis and boundaries are defined and agreed upon with the stakeholders and a rough planning is created. Furthermore repositories are created, docs initialized, etc.

Deadline: 13.4.2018

Expected results:

- Project vision
- Hypothesis
- Project boundaries
- Detail planning
- Basic setup of environment

- Documentation outline

Research

The next step is to deep dive into the problem domain. We read and summarize the important articles about the topic and learn the underlying technologies. Also part of research are interviews.

Deadline: 20.4.2018

Expected results:

- Overview & summary of material: Articles, law, templates, ...
- Knowledge of underlying technologies (Blockchain, Solidity, React, Redux, digital signing, etc.)
- Transcripts of interviews

Prototype

The collected knowledge from the research phase is now transformed into a first prototype which can be challenged by the stakeholders and test users.

Deadline: 27.4.2018

Expected results:

- Prototypes
- Wireframes

Evaluate

Testing & discussing the prototype with the stakeholders and test users give us valuable insights for the further development.

Deadline: 22.6.2018

Expected results:

- Evaluation reports
- User testing reports

Design

Already during the evaluation phase we start the design phase to have an iterative process: The prototype and wireframes are adjusted from the feedback but also the work on the final mockups and software architecture is started.

Deadline 1: 22.6.2018 **Deadline 2**: 27.7.2018

Expected results:

- Mockups
- Software Architecture
- Specifications

Production

Iterative development and finishing a release candidate. This will also take place in combination with design and review phase. Basically in the following loop: Design -> Production -> Review -> Design -> ...

Deadline 1: 29.6.2018 **Deadline 2**: 10.8.2018

Expected results:

- Final product first and second version

Review

Collect feedback of the release candidate, finish documentation and submission of the thesis.

Deadline: 17.8.2018

Expected results:

- Review reports
- Final thesis report

Milestones

17.4.2018 - Prototype & goals specified

- Coach meeting in Brugg
- Specified what's in the prototype and what not
- General project goals specified with Melonport
- Project plan with more detailed & project oriented planning
- Documentation outline fixed

3.5.2018 - Prototype presentation (Meeting in Zug)

- Presentation of the finished prototype in Zug
- All stakeholders invited

5.7.2018 - Presentation of results second iteration (Meeting in Zug)

- Presentation of the results of second iteration in Zug
- All stakeholders invited
- First draft of documentation submitted to coaches
- Finalize specification about final product
  - Requirements fixed
  - All documents available
  - All data available
  - All prototypes ready for extension

6.8.2018 - Presentation of release candidate (Meeting in Zug)

- Presentation of the release candidate in Zug
- Collection of last feedback and adjustments for final release
- All stakeholders invited

16.8.2018 - Final submission

- Official submission of thesis as bound paper
- Final version deployed

Iterations

Iteration 1: Prototype

The first iteration is a functional prototype. The goal is to have something clickable as soon as possible. For this phase, it is not yet important to have all fields and data. But something that can be shown to fund managers to collect first valuable feedback.

Deadline: 27.4.2018

Iteration 2: First draft

The second iteration aims already at the final goal & specification knowing that it is not possible to cover all topics yet. Still, it should be as functional as possbile to gather more detailed user feedback already.

Deadline: 29.6.2018

Iteration 3: Final product

Finally, all feedback is collected and the final product can emerge from the first draft.

Deadline: 10.8.2018

Journal

We send a short status update every week to our stakeholders. They are also stored here for reference.

Calendar Week 8

We started working on the thesis and made a first broad overview over the topics:

- Benjamin started to research into blockchain development particularly Solidity.
- Simon setup the repository and got in touch with possible project supporters from PwC and read a bit into the domain of legal reporting of collective investment schemes.

Calendar Week 9

In the second week we already deep dived into the specific domains:

- Benjamin set up the Solidity development environment according to the Melon setup with dapp.tools, parity dev chain but also looked into truffle suite.
- Simon researched MiFID II and PRIIP and started the glossary for these confusing abbreviations. Furthermore, he finished the strategy part.

Calendar Week 10

This week we had the official kick-off meeting with all stakeholders. See minutes in Appendix.

- Simon updated the project plan according to the feedback and discussed the KIID template provided by PwC with Mona
- Benjamin further read into Solidity, especially data structures like strings and byte64.

Calendar Week 12

A lot of progress is already visible in the different areas:

- Benjamin created a very rough working first draft of the auditing contract
- Simon visualized the logical data structure and started to derive different report requirements according to these structures.

Furthermore, we had our first coach meeting.

Calendar Week 15

- Benjamin started with the auditing interface proposal and it's implementation. Furthermore, he created a rudimentary web interface to this auditing contract.
- Simon refined the documentation outline of the report, created the prototype specification proposal and started to walk through the FINMA KIID document. Furthermore he met with Jenna Zenk and John Orthwein to sync risk management with reporting.

The research takes longer than expected, therefore we jumped forward for the prototype and will implement directly a standard report mockup draft without proper written research foundation. Simon already knows in his head how such a standard report needs to look like. More validating research can also be done later. It is important to have something to show now rather than thorough research.

Risks

Name	Counter measures	Risk (*)
Distractions from classes	Clear timeboxing. Clear planning. Buffers.	2 * 1 = 2
Distractions from job	Clear timeboxing. Clear planning. Clear and upfront communication of availabilities. Support from management.	1 * 1 = 1
Project team out of sync	Weekly team work time slots. Open communication. Weekly reports.	2 * 2 = 4
Dependency on systems out of project scope	Apply subsystem decomposition and isolation techniques from the beginning.	3 * 2 = 6
Losing focus / distraction by details	Weekly reports to coaches and project sponsors to gather feedback.	3 * 2 = 4

(*) Probability of occurrence (1-3) * severity (1-3) = risk

Stakeholders

Melonport (Customer)

University

Coaches

Project team

Fund managers

Regulators

Auditors

Investors

Decisions

Iterations

Prototype

Goals

The purpose of the prototype is to build something early that can already be shown to different stakeholders to collect feedback. This can be in the form of: Click dummies, wireframes, mockups, specifications or actual functional prototypes.

Specification

We suggested the following specifications to the customer and coaches and they approved them (TODO).

Must

These are the minimum tasks that need to be done after the project week.

Melon Auditing Contract Standard Proposal

Introduce new functionality to the community is through a Solidity interface specification proposal. In Ethereum they are called ERC. We create an interface specification proposal for the auditing contract so that the core Melon smart contract developers can see the direction we are going and give early feedback.

- A good example is the ERC-223 Token Standard Proposal
- We will submit this as a MIP (Melon Improvement Proposal)

Standard Report Mockup Draft

To have something to show potential users of the system we create a fictional report that is inspired by the legacy reports that we researched in chapter "Research". This will be in the form of a fictional mockup draft: The deliverable is a static image export from Sketch:

- All applicable information from KIID but also timely reports. E.g. Strategy, SRRI, past performance, benchmark, ...
- All applicable information from the blockchain. E.g. Holdings, recent trades, fund name, risk engineering settings, ...
- Draft means: Not a pixel perfect design, but all information already visible or indicated.
- Fictional: We do not work with real data but we invent data that could be real.

Interchangeable Fund Data Format Draft

Create a general interchangeable fund data format that is the basis for all the different reports and also the data that is actually hashed and signed by the auditing contract.

- A annotated JSON Schema and/or GraphQL Schema
- An example JSON

Bonus

The "must" tasks are estimated pessimistic so there should be time left for some of the following tasks:

Deploy Auditing Contract Draft To Kovan

Deploy a first draft of the auditing contract to Kovan and link it to the current Melon smart contracts.

- Auditing smart contracts coded and ready to deploy
- Deployed to Kovan

Auditing Web Interface Draft

Write a simple web interface to interact with the auditing smart contracts.

Standard Report Web Interface Draft

Start with the web interface that can transform the interchangeable fund data into a standard report.

Implementation

First draft of auditing contract

In a rough first draft for the auditing part of the project, we had the idea of storing the datahash of the audit along with a signature. To proof an audit as legit, the auditor had to create the signature with `web3.eth.sign` [Citation not found] and split it up into the values `r`, `s`, `v`. The auditor then called the function `audit` of the auditing smart contract with the *fund address*, the *datahash* and the values *r*, *s* and *v*. On the Solidity contract, it was a requirement that the provided signature is valid before storing the audit data on the ledger. With the function `checkSignature`, we recovered the datahash and signature values with the elliptic curve recovery function `ecrecover` [4] to the auditor address. This address has to be equal to the sender address.

After receiving feedback from Markus Knecht about this approach, we discussed the necessity of signing the audits "by hand". Basically, a transaction on the blockchain is by itself a signature. The only reasonable use case of an additional signature is when additional participants have to identify themselves. An example for this is the 0x protocol where on-chain data and off-chain data is mixed. This is not the case for auditing (there is only one auditor), so we dropped this functionality for the prototype.

Prototype design

Auditing contract

To receive feedback from the Melonport staff about our prototype implementation of the auditing contract, we set up a MIP (Melon Improvement Proposal)

Test frontend for the auditing contract

Internal

Stakeholder Feedback

Explorative User Testings

Conclusion

First draft

Goals

Implementation

Validative User Testing

Final product

Changes to first draft

Research

Approach:

1. Research & classify available data
2. Research report types (KIID, PRIIPs, etc) and decompose into data requirements
3. Consolidate different data requirements into a general interchangeable fund data format (Github Issue #7)
4. Map available data to interchangeable fund data format.

This is the old world.

SFAMA FINMA KIID Template (German)

The first document we look at is the SFAMA KIID guidelines (in German) which is available on their website and acts as guideline how to compose a KIID to submit to FINMA. It also contains a KIID template on pages 13-14, [2].

Praeamble

Prepended to every KIID document there is the following legal text:

> *Wesentliche Informationen für die Anlegerinnen und Anleger*: Gegenstand dieses Dokuments sind wesentliche Informationen für die Anlegerinnen und Anleger über diese kollektive Kapitalanlage. Es handelt sich nicht um Werbematerial. Diese Informationen sind gesetzlich vorgeschrieben, um Ihnen die Wesensart dieser kollektiven Kapitalanlage und die Risiken einer Anlage zu erläutern. Wir raten Ihnen zur Lektüre dieses Dokuments, sodass Sie eine fundierte Anlageentscheidung treffen können.

Which seems to be the same as in english KIID documents from EFAMA. Here in english:

> *Key information for investors*: This document provides key investor information about this collective investment scheme. It is not marketing material. The information is required by law to help investors understand the nature and the risks of investing in this Fund. Investors are advised to read it so to make an informed decision about whether to invest.

Data classification: off-chain, free text, given

Fund name

> 123 Kollektive Kapitalanlage [, ein Teilvermögen des 123 Umbrella Fonds, Klasse A]

The long name of the fund. E.g. "Credit Suisse (CH) Small Cap Switzerland Equity Fund"

Data classification: on-chain, free text, static

ID/ISIN

> (ISIN: xxxxyyyyyzzz)

The funds ISIN identification. For us this can be the address of the deployed fund contract if fund is not submitted to isin.org.

Data classification: on-chain, generated

Legal entity

> Fondsleitung: ABC Fondsleitung (Schweiz) AG [, eine Gesellschaft ABC Gruppe]

Data classification: off-chain, free text, static

Objectives and investment policy

> Anlageziele und Anlagepolitik

Asset categories

> Hauptkategorien der für die Anlage in Frage kommenden Finanzinstrumente;

A Melon fund selects its manageable assets during setup and they can't change anymore later. So this can be displayed as a simple list.

Data classification: on-chain, static, configuration

Participation

> Hinweis auf das Kündigungsrecht des Anlegers unter Angabe der Rücknahmefrequenz

Generally, Melon funds do not restrict investors from withdrawing funds or termination. But every fund can have a separate Participation/compliance module (see Melon smart-contracts: ComplianceInterface) which can restrict investing & redemption through boolean functions with the following inputs: `investor`, `shareQuantity`, `giveQuantity`. Therefore participation modules can restrict withdrawal according to liquidity, ...

The redemption of shares is also restricted if the fund does not hold enough of the quote asset (i.e. cash, e.g. ETH or DAI). That said, it is always possible to redeem in slices (emergency redeem: Receive the managed assets directly in proportion of shares).

Data classification: on-chain, static, configuration

Data representation: Common participation modules could be described with free text.

Strategy/Objectives

> Angabe, ob die kollektive Kapitalanlage ein bestimmtes Ziel in Bezug auf einen branchen-spezifischen, geografischen oder anderen Marktsektor oder in Bezug auf spezifische Anlageklassen verfolgt;

This is just free text describing the strategy/objectives of the fund. For a manual audit, it should be easy to verify if the objectives are followed or not.

Data classification: off-chain, free text

Open questions

- What are examples of strategy/objectives?
- How are those currently enforced?
- What happens if a fund manager violates these objectives?
- Example: Credit Suisse (CH) Small Cap Switzerland Equity Fund: Invests in small titles in Switzerland.
  - What happens if the fund invests in a company not listed in Switzerland?
  - What happens if a company that this fund is invested in is bought by an US company?

Automation

> Angabe, ob die kollektive Kapitalanlage die Anlageentscheide nach freiem Ermessen treffen kann und ob ein Referenzwert herangezogen wird;

Note: It also talks about benchmarks or reference prices but we will go into this topic later.

A Melon fund could be automatically managed by a bot similar to an ETF which is currently not visible on-chain. In the future it might be possible to store some strategies on-chain but for this thesis we stick to the status quo: A fund can indicate automation just by describing it with free text. An auditor might check a fund for automation which did not declare its automation or check if automation was executed as described.

Data classification: off-chain, free text.

Income distribution

> Angabe, ob die Erträge der kollektiven Kapitalanlage ausgeschüttet oder thesauriert werden.

Not applicable. An investor always holds a percentage of the AUM in Melon. There is no concept of dividends or similar.

Data classification: Not applicable

Leverage, Hedging, Arbitrage, Dept Securities

> Werden besondere Anlagetechniken verwendet, wie z.B. „hedging“, „arbitrage“ oder „leverage“, so sind deren Auswirkungen auf die Wertentwicklung der kollektiven Kapitalanlage anzugeben; Investiert die kollektive Kapitalanlage in Schuldtitel, sind diese zu beschreiben;

The Melon smart contracts by itself do currently not actively support leverage, hedging, etc. That said, it is theoretically possible to buy into an asset/token which could be leveraged. E.g. Lendroid, Salt lending, ...

Risk assessment is done per asset.

Data classification: Not applicable, free text

Asset selection criteria

> Nach welchen Kriterien die Anlagen ausgewählt werden, z.B. nach „Wachstum“, „Wert“ oder „hohe Dividenden“ mit Erläuterung dieser Kriterien;

In the current smart contract implementation, the asset universe is predefined by the PriceFeed: A fund cannot invest in an asset which is not prelisted on the general pricefeed. Assets are not categorized.

Data classification: N/A yet, free text

Open questions

- Will assets on the blockchain be categorized in the future so that a fund can set a asset selection critera as smart contract? E.g. A fund that only invests in renewable energy cannot buy a token representing a share on a atomic power plant.

Factors influencing performance

> bei strukturierten kollektiven Kapitalanlagen die Faktoren, welche die Ausschüttungen sowie die Wertentwicklung der kollektiven Kapitalanlage beeinflussen;

There are no other factors influencing performance than the actual performance of the fund.

Data classification: Not applicable, given

Transaction costs

> dass die Transaktionskosten zu Lasten des Fondsvermögens gehen und somit den Ertrag der kollektiven Kapitalanlage schmälern;

Transaction costs (i.e. gas) are paid by the fund manager in the current implementation. It is her responsibility to set the fees in a way that the transaction costs are paid. In future versions we might implement a feature which deducts transaction costs directly from the fund performance.

Data classification: Not applicable, given

Minimum holding period

> Sofern im Prospekt oder anderweitig eine Mindesthaltedauer empfohlen wird, ist der folgende Wortlaut beizufügen: „Empfehlung: Diese kollektive Kapitalanlage ist unter Umständen für Anlegerinnen und Anleger nicht geeignet, die ihr Geld innerhalb eines Zeitraumes von […] aus der kollektiven Kapitalanlage wieder zurückziehen wollen.“;

There is no concept of minimum holding period in the current implementation. That said, it might be part of the managers strategy to recommend something or enforces it through a participation module.

Data classification: free-text, (on-chain, static, configuration through a participation module)

Funds of funds

> Bei Dachfonds ist anzugeben, nach welchen Kriterien die Zielfonds ausgewählt werden.

It is currently forbidden to create fund of funds.

Data classification: Not applicable

Risk profile

Risk indicator

Comparable indicator between 1 (low risk/low reward) to 7 (high risk/high reward). The SFAMA document [2], pages 16ff describes how to calculate this indicator. It is the same as ESMA's SRRI calculation. It is mainly based on overall volatility of a fund.

Data classification: Complicated, historic, on-chain

Given explanations

There are some given explanations:

> • dass der Indikator keine zuverlässige Aussage über die zukünftige Wertentwicklung enthält; • dass die Risikokategorie Veränderungen unterliegen und über die Jahre variieren kann; • dass die geringste Risikokategorie nicht einer risikofreien Anlage entspricht;

Which are just warnings about risks in general.

Data classification: Given

Timely reports

MiFID II and PRIIP

Melon Risk Engineering

uPort

Solidity research

Resources

Solidity

Solidity is a programming language for smart contract development on the Ethereum blockchain.

Parity

Parity is a client to interact with the Ethereum blockchain. It provides an easy way to set up a development chain where we can test our own smart contracts without limitations.

Web3.js

Web3.js is a JavaScript API for interacting with local or remote ethereum nodes. It enables us to develop clients which communicate with our smart contracts on the blockchain.

Learnings

While learning *Solidity*, we created a cheatsheet to have a compact summary while developing our own smart contracts. The cheatsheet can be found in the appendix Solidity cheatsheet.

The *ABI* (Application Binary Interface) can be understood as the API to a deployed smart contract. *Web3.js* or a similar library uses this interface to interact with the Ethereum ecosystem.

Web3 requires us to specify a provider. Here we point to a running Ethereum client, in our case a running *Parity* instance. Through the ABI, we can call functions on the contract instance through `contract.methods`. Methods that store value on the blockchain can be executed by `send()`, Read-only methods can be called with `call()`. The result is a JSON RPC response containing the requested values. It is possible to catch the `error`, `transactionHash` or `receipt` with the web3-function `on()`. All events on the chain can be retrieved with the `getPastEvents()` method. We could also subsrcibe on new events and get notified instantly when they are fired from withing smart contracts.

Assessment

The *melon protocol* is already dependent on the Ethereum blockchain and uses smart contracts written in Solidity extensively. Furthermore, smart contracts support us to store melon fund audit information in a secure and consistent way and provide us with the features we require to verify auditors.

We also looked into truffle suite and dapp tools. *Parity* and *web3.js* provide us with all the features we need to develop smart contracts and interact with them for now though.

Possible limitations and considerations

It is of upmost importance that the smart contracts we develop for this project are as light as possible. Storing data on the blockchain can be expensive. We presume that it is in the interest of the auditors that the fee of storing audit information of a fund is as small as possible. For this, we will keep an eye on the *gas cost* that is required to store information.

As this project and the corresponding smart contracts are not dealing with *Ether* directly, we do not have to account security considerations as in sending or receiving currencies. Most of the data that we will be acquiring from the blockchain can be regarded as *read only*.

Solidity Cheat Sheet

This is a summary of Solidity In Depth.

Layout of source files

Versions

```
pragma solidity ^0.4.0;
```

`^` means "only works with compilers 0.4.x"

We will use at least version *0.4.20* for our contracts.

Import

```
import "filename"; // import from 'global' or same directory
import * as symbolName from "filename";
import "filename" as symbolName; // same as above
import {symbol1 as alias, symbol2} from "filename";

import "./x" as x; // strictly import from same directory
```

Path prefix remapping

If we clone github.com/ethereum/dapp-bin/ locally to /usr/local/dapp-bin, we can use:

```
import "github.com/ethereum/dapp-bin/library/iterable_mapping.sol" as it_mapping;
```

And run the compiler with:

```
solc github.com/ethereum/dapp-bin/=/usr/local/dapp-bin/ source.sol
```

Comments

```
// single line

/*
    Multiline
*/
```

Natspec comments

```
/// single line
```

```
/** multiline */
```

Example:

```
/** @title Shape calculator. */
contract shapeCalculator {
    /** @dev Calculates a rectangle's surface and perimeter.
    * @param w Width of the rectangle.
    * @param h Height of the rectangle.
    * @return s The calculated surface.
    * @return p The calculated perimeter.
    */
    function rectangle(uint w, uint h) returns (uint s, uint p) {
        s = w * h;
        p = 2 * (w + h);
    }
}
```

Structure of a contract

State Variables

Permanently stored in contract storage.

```
unit storedData;
```

Functions

```
function bid() {...}
```

Function modifiers

Amend the semantics of a function, mostly used for require.

Declaration:

```
modifier onlySeller() { ... }
```

Usage:

```
function abort() onlySeller { ... }
```

Events

Interfaces for EVM logging

Declaration:

```
event HighestBidIncreased(address bidder, uint amount);
```

Trigger:

```
emit HighestBidIncreased(msg.sender, msg.value);
```

Struct Types

Group several variables

```
struct Voter {
    uint weight;
    bool voted;
    address delegate;
    uint vote;
}
```

Enum Types

```
enum State { Created, Locked, Inactive }
```

Types

Value Types

Booleans

```
bool t = true;
bool f = false;
```

Operators: !, &&, ||, ==, !=

Integers

Aliases for int256 and uint256:

```
int i = -1;
uint j = 1;
```

Declare size from `(u)int8` to `(u)int256`, e.g.:

```
int24 max = 8388608;
```

Comparisons: <=, <, ==, !=, >=, > Bit operators: &, |, ^, ~ Arithmetic: +, -, *, /, %, *, <<, >>

Fixed point numbers

They are not fully supported in Solidity yet, they can only be declared.

Address

20 byte value. Operators: <=, <, ==, !=, >=, >

Members:

Query the **balance** of an address:

```
uint b = a.balance;
```

Send ether in units of wei to an address:

```
a.transfer(10); // transfers 10 wei to address a
```

If the execution fails, the contract will stop with an exception If a is a contract address, its code will be executed with the transfer call.

Low level counterpart of transfer (returns 'false' on fail):

```
bool success = a.send(10);
```

NOTE: send is dangerous, use transfer or the withdraw pattern.

call, callcode, delegatecall: interface with contracts that do not adhere with the ABI.
NOTE: Only use as last resort, they break the type-safety of Solidity. Arguments of call are padded to bytes32 type. Returns bool that indicates if the function terminated (true) or threw an exception (false).

```
address nameReg = 0x72ba7d8e73fe8eb666ea66babc8116a41bfb10e2;
nameReg.call("register", "MyName");
nameReg.call(bytes4(keccak256("fun(uint256)")), a); // function signature
```

Adjust the supplied gas with .gas():

```
namReg.call.gas(1000000)("register", "MyName");
```

Control the supplied ether value:

```
nameReg.call.value(1 ether)("register", "MyName");
```

Combine both:

```
nameReg.call.gas(1000000).value(1 ether)("register", "MyName");
```

Query the **balance of the current contract**:

```
this.balance;
```

Fixed-size byte arrays

`bytes1` , `bytes2` , `bytes3` , up to `bytes32`
`bytes` is an alias for `bytes1` .

Comparisons and bit operators can be used like on ints.

Index access:

```
bytes2 b2 = "hi";
byte b1 = b2[0]; // access first byte, read only!
```

Length:

```
b1.length; // returns the fixed length of the byte array
```

`bytes` and `string` are not value types!

Rational and integer literals

Decimal fraction literals:

```
1.
.1
1.3
```

Scientific notation is supported:

```
2e10
-2e10
2e-10
2.5e1
```

Division on integers converts to a rational number, but it cannot be stored in any way yet.

```
int i = 5 / 2; // this throws a compiler error
int j = 6 / 2; // this works
```

String literals

```
"use double quotes"
'or single quotes'
```

They are implicitly convertible to `bytes1` ... `bytes32` **if they fit**.

Escape characters are possible:

```
\n // and the like
\xNN // hex
\uNNNN // unicode
```

Hexadecimal literals

```
hex"001122FF"
```

The content must be a hexadecimal string. The value will be the binary representation.. They behave like string literals.

Enums

```
enum ActionChoices { GoLeft, GoRight, GoStraight, SitStill }
```

```
ActionChoices choice = ActionChoices.GoStraight;
```

Function types

internal functions
Can only be called inside the current contract. This is the *default* for functions.

Use internal functions in **libraries**:

```
library SomeLibrary {
    // declare internal functions
}

contract SomeContract {
    using SomeLibrary for *;
    // use internal functions
}
```

external functions
Consist of an address and a function signature.

Use external functions between **contracts**:

```
contract Oracle {
    // define external functions
}

contract OracleUser {
    Oracle constant oracle = Oracle(0x1234567); // known contract
    oracle.query(...);
}
```

Notation:

```
function (<parameter types>) {internal|external} [pure|constant|view|payable] [returns (<return types>)]
```

Access the function type:

```
f // call by name -> internal function
this.f // -> external function
```

Return the ABI function selector:

```
this.f.selector; // type: bytes4
```

Reference Types

Reference types have to be handled more carefully, because storing them in **storage** is expensive.

Data location

Complex types (arrays & structs) have a data location (`storage` or `memory`).

- Default for function parameters is `memory`
- Default for local variables is `storage`

- `storage` is forced for state variables

There is a third location `calldata` where function arguments are stored.

Arrays

Can have a fixed size or can be dynamic.

Array of 5 dynamic arrays of uint:

```
uint[][5] a;
uint a32 = a[2][3]; // access second uint in third array
```

NOTE: notation is reversed!!!

USEFUL - convert from string to bytes:

```
string memory s = "hello world";
b = bytes(s);
```

Create a **getter** for arrays automatically by marking them `public`:

```
int32 public intArray;
```

But values can only be obtained with a numeric index.

Allocating Memory Arrays

Use `new` for arrays with variable length in *memory*.

```
uint[] memory a = new uint[](7);
bytes memory b = new bytes(7);
```

Array Literals / Inline Arrays

Arrays written as an expression:

```
[uint(1), 2, 3]; // evaluates to a 'uint8[3] memory' array
```

The cast on the first element is necessary to define the type. They cannot be assigned to dynamic arrays at the moment.

Members

- length
  Dynamic arrays can be resized in *storage* with `.length`.
  The size of *memory* arrays is fixed once they are created.
- push
  Use `push` to append an element on dynamic storage arrays and `bytes`.

NOTE:
It is not possible to return dynamic content from external function calls. The only workaround now is to use large statically-sized arrays.

Structs

```
struct Funder {
    address addr;
    uint amount;
}

Funder f1 = Funder({addr: msg.sender, amount: msg.value}); // create with names
Funder f2 = Funder(msg.sender, msg.value); // simple create
f1.amount += msg.value; // access
```

Mappings

Formal definition:

```
mapping(_KeyType => _ValueType)
```

- Keytype can be almost anything except for a mapping, dynamically sized array, contract, enum or struct.
- ValueType can be anything, even another mapping.

The key data is not actually stored in a mapping, only its `keccak256` hash. Mappings do not have a length. Mappings are only allowed for **state variables** (or storage reference types in internal functions).

The only way to retreive a value from a mapping is by its key. Mappings are not enumerable.

Example:

```
// declaration
mapping(address => Voter) public voters;

// change a value in a mapping
chairperson = msg.sender;
voters[chairperson].weight = 1;
```

LValue Operators

```
a += e;
a -= e;
a *= e;
a /= e;
a %= e;
a |= e;
a &= e;
a ^= e;
a++;
a--;
++a;
--a;
```

delete

`delete` assigns the initial value for the type of a:

```
int i = 42;
delete i; // i is now 0
```

Delete on dynamic arrays assigns an array of size 0.
Delete on static arrays resets all values.
Delete on structs resets all values.

Conversions

Implicit conversions

Possible when no information is lost:

- `uint8` to `uint17` is possible
- `int128` to `int256` is possible
- `int8` to `uint256` is NOT possible
- uints can be converted to bytes of the same size or larger
- `uint160` can be converted to `address`

Explicit conversions

Use with care!

```
int8 y = -3;
uint x = uint(y);
```

```
uint32 a = 0x12345678;
uint16 b = uint16(a); // b will be 0x5678 now -> information loss
```

Type Deduction (var)

It is not necessary everytime to assign a type.

```
uint24 x = 0x123;
var y = x; // y has type uint24 automatically here
```

Units and global variables

Ether units

Possible ways to work with ether units:

- Literal number with suffix (`wei` , `finney` , `szabo` , `ether`)
- Literal number without suffix is always `wei`

Calculation works as expected.

```
2 ether == 2000 finney // evaluates to true
```

Time units

Calculation works as expected.
The base unit is *seconds*.

```
1 == 1 seconds
1 minutes == 60 seconds
1 hours == 60 minutes
1 days == 24 hours
1 weeks == 7 days
1 years == 365 days
```

The suffixes cannot be applied to variables. Do it like this:

```
uint daysAfter = 42;
if (now >= daysAfter * 1 days) {...};
```

Special variables and functions

Block and transaction properties

- `block.blockhash(uint blockNumber)` returns (`bytes32`): hash of the given block - only works for 256 most recent blocks excluding current
- `block.coinbase` (`address`): current block miner's address
- `block.difficulty` (`uint`): current block difficulty
- `block.gaslimit` (`uint`): current block gaslimit
- `block.number` (`uint`): current block number
- `block.timestamp` (`uint`): current block timestamp as seconds since unix epoch
- `gasleft()` returns (`uint256`): remaining gas
- `msg.data` (`bytes`): complete calldata
- `msg.sender` (`address`): sender of the message (current call)
- `msg.sig` (`bytes4`): first four bytes of the calldata (i.e. function identifier)
- `msg.value` (`uint`): number of wei sent with the message
- `now` (`uint`): current block timestamp (alias for block.timestamp)
- `tx.gasprice` (`uint`): gas price of the transaction
- `tx.origin` (`address`): sender of the transaction (full call chain)

NOTE: `now` is just the timestamp from the block!

Error handling

```
assert(bool condition) // for internal errors
require(bool condition) // for errors in inputs or external components
revert() // abort execution, revert state changes
```

Mathematical and cryptographic functions

```
addmod(uint x, uint y, uint k) returns (uint) // compute (x + y) % k
mulmod(uint x, uint y, uint k) returns (uint): // compute (x * y) % k
keccak256(...) returns (bytes32) // compute Ethereum-SHA-3 hash of args
sha256(...) returns (bytes32) // compute SHA-256 hash of args
sha3(...) returns (bytes32) // alias to keccak256()
ripemd160(...) returns (bytes20) // compute RIPEMD-160 hash of args
ecrecover(bytes32 hash, uint8 v, bytes32 r, bytes32 s) returns (address) // elliptic curve signature
```

The arguments are packed without padding.
ecrecover example

Address related

```
<address>.balance (uint256) // balance of the address in wei
<address>.transfer(uint256 amount) // send amount of wei to address, throws on failure
<address>.send(uint256 amount) returns (bool) // send amount of wei to address, returns false on failure
```

Contract related

```
this // the current contract, convertible to address
selfdestruct(address recipient) // destroy current contract, send funds to address
```

Expressions and control structures

Input parameters

```
function taker(uint _a, uint _b) public pure {
    // do something with _a and _b.
}
```

Output parameters

Returning multiple values is possible.

```
function arithmetics(uint _a, uint _b)
    public
    pure
    returns (uint a, uint b)
{
    a = 1;
    b = 2;
}
```

Return parameters are initialized to zero.

Control structures

They can be used the same as in C or Javascript:

- `if`
- `else`
- `while`
- `do`
- `for`
- `break`
- `continue`
- `return`

- `? :`

There is no type conversion from non-boolean to boolean (`1` is not `true` !).

Function calls

Internal function calls

Internal functions (from the same contract) can be used recursively.

External function calls

```
this.g(8);
c.g(8) // where c is a contract instance
```

Amount of wei and gas can be specified when calling functions from other contracts:

```
contract InfoFeed {
    function info() public payable returns (uint ret) { return 42; }
}

contract Consumer {
    InfoFeed feed; // contract instance
    function callFeed() public { feed.info.value(10).gas(800)(); }
}
```

`payable` must be used for `info()` to have the `.value()` option.

WARNING:
Called contracts can change state from our own contracts. Write functions in a way that calls to external functions happen after any changes to state variables in our contract so our contract is not vulnerable to a *reentrancy exploit*.

Named calls

Enclose args in `{}` , then the order doesn't matter:

```
f({value: 2, key: 3});
```

Creating contracts with new & constructors

Contracts can create other contracts with `new` :

```
contract D {
    uint x;
    function D(uint a) public payable { // ctor
        x = a;
    }
}

contract C {
    D d = new D(4); // will be executed as part of C's constructor

    function createD(uint arg) public {
        D newD = new D(arg);
    }
```

```
}
```

NOTE: Creating constructors by their name is deprecated. This is the new, preferred way:

```
    constructor() public payable {
        //
    }
```

Assignment

Tuple syntax is possible:

```
function f() public pure returns (uint, bool, uint) {
    return (7, true, 2);
}

var (x, b, y) = f(); // specifying types is not possible here
(x, y) = (2, 7); // assign to predefined variables
(x, y) = (y, x); // swap
(data.length,) = f(); // rest of the values can be ignored (returns 2 values but we only care about first)
(,data[3]) = f(); // ignore beginning values
(x,) = (1,); // one component tuple
```

Scoping and declarations

For version 0.4.x, a variable declared anywhere in a function is available everywhere in the function (like Javascript). In version 0.5.x, this will change.

Error handling (Assert, Require, Revert, Exceptions)

Solidity uses state-reverting exceptions.

- Use `assert` to check invariants
- Use `require` to ensure input values
- Use `revert` to throw an exception, revert the current call (and done subcalls)

Catching exceptions is not yet possible.

`assert` will use all gas, `require` uses none.

Contracts

Creating Contracts

With web3js: web3.eth.Contract

Only one constructor is allowed --> ctor overloading is not possible. Cyclic dependencies between contracts are not possible.

Visibility and getters

There are two kinds of function calls in Solidity, so there are four types of **visibilities for functions**.

external

Called via other contracts and transactions.

To call it from inside a contract, we have to use `this.f()`.

public (default)

Call internally or via messages. For public state variables, a getter is generated automatically.

internal

Functions and state variables can only be accessed from within the *contract (or deriving contracts)*.

private

Functions and state variables can only be accessed from within the *contract*.

NOTE: private stuff is still visible for everyone, just not accessible!

Getter functions

The compiler automatically creates getter-functions for public state variables.

This:

```
contract Complex {
    struct Data {
        uint a;
        bytes3 b;
        mapping (uint => uint) map;
    }
    mapping (uint => mapping(bool => Data[])) public data;
}
```

will generate the following function:

```
function data(uint arg1, bool arg2, uint arg3) public returns (uint a, bytes3 b) {
    a = data[arg1][arg2][arg3].a;
    b = data[arg1][arg2][arg3].b;
}
```

Function modifiers

Modifiers can change the behaviour of functions. The function body is inserted where the special symbol _ appears.

```
modifier onlyOwner {
    require(msg.sender == owner);
    _;
}

function changePrice(uint _price) public onlyOwner {
    price = _price;
}
```

Multiple modifiers can be used in a whitespace-separated list. All symbols visible in the function are visible for the modifier.

Constant state variables

State variables can be declared as `constant`. Then they have to be assigned from an expression which is a constant at compile time.

These functions are allowed:

- `keccak256`
- `sha256`
- `ripemd160`
- `ecrecover`
- `addmod`
- `mulmod`

The only supported types valid for now are **value types** and **strings**.

Funtions

View functions

Promise **not to modify state**. Can be declared with `view`.

These things are considered to modify state:

- Writing to state variables
- Emitting events
- Creating other contracts
- Using `selfdestruct`
- Sending ether
- Calling functions not marked `view` or `pure`
- Using low-level calls
- Using inline assembly with opcodes

NOTE: getter methods are marked `view`.

Pure functions

Functions that do **not read from or modify the state**.

These things are considered to read from state:

- Reading from state variables
- Accessing `this.balance` or `<address>.balance`
- Accessing `block`, `tx` or `msg`
- Calling functions not marked with `pure`
- Inline assembly with opcodes

Fallback function

The unnamed function. This is called when no other function matches the function identifier.

Sending ether to this contract will cause an exception (no other functions are defined):

```
uint x;
function() public { x = 1; }
```

If ether is sent to this contract, there is no way to get it back:

```
function() public payable { }
```

Function overloading

Function overloading is possible (but not for ctors).

```
contract A {
    function f(uint _in) public pure returns (uint out) {
        out = 1;
    }

    function f(uint _in, bytes32 _key) public pure returns (uint out) {
        out = 2;
    }
}
```

If there is not exactly one candidate for the function, resolution fails. For example: `f(unit8)` and `f(uint256)` fails when `f` is called with a `uint8` value or below.

Events

The "logging" mechanism of ethereum.

SPV proofs are possible: If an external entity supplies a contract with an SPV proof, it can check that the log actually exists in the blockchain (but block headers have to be supplied).

Up to three arguments can receive the attribute `indexed`. We can then search for these arguments. Indexed arguments will not be stored themselves, we can only search for these values. If arrays are used as indexed arguments, their `keccak256` hash will be stored.

With `anonymous`, the signature of the event is not stored. All non-indexed arguments will be stored in the data part of the log.

Event example:

```
contract ClientReceipt {
    event Deposit(
        address indexed _from,
        bytes32 indexed _id,
        uint _value
    );

    function deposit(bytes32 _id) public payable {
        emit Deposit(msg.sender, _id, msg.value); // 'Deposit' is now filterable with JS
    }
}
```

Look for events with *javascript*:

```
var abi = /* abi as generated by the compiler */;
var ClientReceipt = web3.eth.contract(abi);
var clientReceipt = ClientReceipt.at("0x1234...ab67" /* address */);

var event = clientReceipt.Deposit();

// watch for changes
event.watch(function(error, result){
    // result will contain various information
    // including the argumets given to the `Deposit`
    // call.
    if (!error)
        console.log(result);
});

// or callback to start watching immediately
var event = clientReceipt.Deposit(function(error, result) {
    if (!error)
        console.log(result);
});
```

There is also a **low-level interface** for logs.

Inheritance

Solidity supports multiple inheritance. All functions are *virtual* (most derived function is called).

The code from inherited contracts is copied into one contract.

Use `is` to derive from another contract.

If a contract doesn't implement all functions, it can only be used as an interface:

```
function lookup(uint id) public returns (address adr); // 'abstract' function
```

Multiple inheritance is possible:

```
contract named is owned, mortal { ... }
```

Functions can be overridden by another function with the same name and the same number/types of inputs.

If the constructor takes an argument, it must be provided like this:

```
contract PriceFeed is named("GoldFeed") { ... }
```

To specifically access functions from base contracts, use `super`:

```
contract Base is mortal {
    function kill() public { super.kill(); }
}
```

Constructors

Constructor functions can be `public` or `internal`.

```
contract B is A(1) {
    function B() public {}
```

```
}
```

An `internal` ctor marks the contract as abstract!

Arguments for base constructors

```
contract Base {
    uint x;
    function Base(uint _x) public { x = _x; }
}

contract Derived is Base(7) {
    function Derived(uint _y) Base(_y * _y) public {
    }
}
```

Abstract contracts

Contracts where at least one function is not implemented.

This is a function declaration:

```
function foo(address) external returns (address);
```

Careful: this is a function type (variable which type is a function):

```
function(address) external returns (address) foo;
```

Interfaces

- Cannot have any functions implemented
- Cannot inherit other contracts or interfaces
- Cannot define variables
- Cannot define structs
- Cannot define enums

Use keyword `interface`:

```
interface Token {
    function transfer(address recipient, uint amount) public;
}
```

Libraries

Libraries are assumed to be stateless. They are deployed only once at a specific address.

Restrictions in comparison to contracts

- No state variables
- Cannot inherit or be inherited
- Cannot receive ether

Example:

```
library Set {
  // type of Data is 'storage reference', so only the storage address, not the content is saved here
  struct Data { mapping(uint => bool) flags; } // will be used in the calling contract!

  function insert(Data storage self, uint value)
      public
      returns (bool)
  {
      if (self.flags[value])
          return false; // already there
      self.flags[value] = true;
      return true;
  }
  // ...

contract C {
    Set.Data knownValues;

    function register(uint value) public {
        // library functions can be called without a specific instance!
        // the 'instance' is the current contract...
        require(Set.insert(knownValues, value));
    }
```

Using for

Attach library functions to any type:

```
using A for B; // where A is the library and B the type
```

With the example from above:

```
contract C {
    using Set for Set.Data; // this is the crucial change
    Set.Data knownValues;

    function register(uint value) public {
        // Here, all variables of type Set.Data have
        // corresponding member functions.
        // The following function call is identical to
        // `Set.insert(knownValues, value)`
        require(knownValues.insert(value));
    }
}
```

We can also extend elementary types:

```
using Search for uint[];
```

Data Classification

Reasoning

CAP theorem

The Ethereum blockchain does not behave like a typical distributed data store in the classical sense. Nonetheless, we want to discuss the CAP theorem and its guarantees here.

Source: Wikipedia

Consistency

> Every read receives the most recent write or an error

Regarding On-Chain data for the reports, this is not relevant. We generate reports in regards to a timespan. This means that we do not have to depend on values being the "most recent".

Availability

> Every request receives a (non-error) response – without guarantee that it contains the most recent write

Again - with On-Chain data, we work with data retrieved from a timespan. Availability is not an issue.

Partition tolerance

> The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes

As every node in the blockchain eventually works with the same ledger, partition tolerance is not an issue.

ACID

Atomicity

Consistency

Isolation

Durability

Eventual consistency

BASE (Basically Available, Soft state, Eventual consistency)

The Ethereum blockchain does NOT follow *eventual consistency*, but *strong consistency*. Source: Bitcoin Guarantees Strong, not Eventual, Consistency

Our approach

Minutes

Coachmeeting 2018-04-17

Present:

- Simon Emanuel Schmid
- Benjamin Zumbrunn
- Sarah Hauser (FHNW)
- Markus Knecht (FHNW)

Agenda

- Present and discuss documentation outline
- Discuss and approve prototype specification
- Discuss and approve project goals
- Discuss more detailed project plan (Github)
- Possible collaboration for the Melon Hackathon
- Citations?

Documentation outline

The content of the report is good, but the structure is not as intended. Write as if the project is finished, e.g. do not include something like a step by step report. Place planning, iterations, milestones, variants, prototype etc. on the end of the document.

- Research -> State of the art. Or choose a different term, Ms. Hauser will supply one
- Delete conclusion & approach
- New chapter Analysis: summary of what we learned from the "research",
- Evaluate chapter: in appendix
- Design + Production -> new Chapter Solution

We propose a new outline via mail.

Protoype specification

The MIP is a good idea, but we should not focus on creating the process to define such MIPs (do not waste time...). Document the concept of *Contract Standard Proposal* in the chapter *state of the art*. Standard Report Mockup Draft is a good idea. Interchangable Fund Data Format.

Think about what we sign --> only the raw data (Melon Report)? An audit about the data?

Melon report: The JSON representing the factsheet must be constructed to also be extensible for KIID, PRIIPS etc. It would be really nice to have a JSON that is also extensible for the "new world".

Knecht: We should not care about security considerations about signing (e.g. do I sign the data I have just seen?), but care about signing on protocol level.

Knecht: it is important that we work with ONE specific version of the protocol, e.g. the state of the competition in February(?).

Important: Define list of use cases, actors (regulators etc.).

Project goals

Before specifying project goals (deliverables), define what a user can do with our project. New milestone for 3.5.2018: Use cases are defined and agreed on.

Define the goals from the *user perspective*, **not** from the technical perspective.

Show the current project goals in the meeting with Melonport as *ideas*. Knecht: Specify the *what*, not the *how*.

Detailed project plan (GitHub)

Do not include the milestones in the "Kanban" board.

Melon Hackathon

Simon will write Mr. Knecht about the hackathon.

Citations

Use the IEEE format (with numbers), maybe there is a plugin. Ms. Hauser sent a mail with a link. Maybe use LaTeX when citation does not work at all.

Expert

The expert for the project will be Konrad Durrer (CSS). He organizes a midterm presentation in May.

Coachmeeting 2018-03-27

Present:

- Simon Emanuel Schmid
- Benjamin Zumbrunn
- Sarah Hauser (FHNW)
- Markus Knecht (FHNW)

Agenda

- Present updated project plan (3 iterations)
- Discuss current state:
  - State of research
  - State of solidity
- Fix date for thesis defense

Bachelor thesis defence

Accoring to Ms. Hauser, our defence can be held in the first week of the presentations: September 3rd to 7th.

Project plan

There are some mistakes on the project plan:

- prototype date is wrong -> 19.4.
- first iteration -> should be second iteration

For 19.4.: attach to milestone: fix project plan with more details

For 17.4.: send coaches a draft of the documentation (list of contents is fixed)

After the first draft of the product, the first draft of documentation will be reviewed by coaches (5.7.).

For 5.7.: everything about the final product must be decided

- requirements fixed
- all documents available
- all data available
- all prototypes ready for extension

State of research

Simon went through the research chapter with the coaches and discussed what he is working on right now.

Decide soon: which market will the product be for? (europe/swiss) Hauser: do not solve the problem of "different markets"

Define exact terms for the "technical" world and the "customer" world that will be used througout the project.

For the risk profile: Suggestion: if we cannot calculate the "monte carlo" risk, we should show that it is possible with the data that we have.

State of solidity

Benjamin showed the draft of the auditing contract prototype and discussed the result with the coaches.

Notes from Mr. Knecht:

- bytes32 signature to bytes32 rs components doesn't make sense (information loss)
- Assembly code is critical: maybe deliver rsv of the signature directly
- Check if delivering a signature is really important. (The 'sender' of the audit already delivers a signature with his address)
- Define prototype specification with use cases.

Kick-off meeting 2018-03-07

Present:

- Simon Emanuel Schmid
- Benjamin Zumbrunn
- Mona El Isa (Melonport)
- Reto Trinkler (Melonport)
- Sarah Hauser (FHNW)
- Markus Knecht (FHNW)

FHNW

- Ms. Hauser is away for the whole month of may
- Always send protocols from meetings to both coaches
- Always send decisions about the project (with melonport) to both coaches

Thesis release

- A poster and an interactive demo is required
- Two printed versions of the thesis must be provided for the FHNW
- Send in a draft of the thesis as soon as possible for review
- Plan three iterations of work/evaluate
- Define milestones and give more detail on planned work

Project

We might be able to assemble the benchmark data from coinmarketcap or similar platforms.

Our Gitbook solution for project reporting is agreed by the coaches.

Possible verification of the generated reports:

- History of all concerning trades
- Hyperlinks which point directly to the blockchain on IPFS (see IPFS Mesh)

Dates

- We will send a doodle with tuesday-dates (3pm-6pm) for coach reviews at Campus Brugg/Windisch

Progress Meetings in Zug

We already fixed the following dates for progress meetings with all stakeholders. They will happen in the Melonport Office in Zug.

- May 3, 14:00: Presentation of the prototype
- July 5, 14:00: Discuss Design & Architecture before final sprint
- August 6, 14:00: Last meeting before submission

References

1	SFAMA, Vorlage für das KIID für Effektenfonds sowie für übrige Fonds für traditionelle Anlagen in der Form von Publikumsfonds (gemäss SFA-KIID-Richtlinien), *Swiss Funds & Asset Management Association*, 2012.
2	SFAMA, Richtlinien zu den „Wesentlichen Informationen für die Anlegerinnen und Anleger“ für Effektenfonds sowie für übrige Fonds für traditionelle Anlagen in der Form von Publikumsfonds , *Swiss Funds & Asset Management Association*, 2012.
3	Ethereum Foundation, web3.eth.sign, 2018.
4	Ethereum Foundation, Units and Globally Available Variables - Mathematical and Cryptographic Functions, 2018.

AUM

Assets under management. A number indicating the total value of all assets in a fund denominated in the funds quote asset.

CISA

(FINMA)

Collective Investment Schemes Act / Kollektivanlagengesetz, KAG

https://www.admin.ch/opc/en/classified-compilation/20052154/index.html

CISO

Collective Investment Schemes Ordinance / Kollektivanlagenverordnung, KKV

https://www.admin.ch/opc/en/classified-compilation/20062920/index.html

CISO-FINMA

Ordinance of the Swiss Financial Market Supervisory Authority on Collective Investment Schemes / Kollektivanlagenverordnung-FINMA, KKV-FINMA

https://www.admin.ch/opc/en/classified-compilation/20140344/index.html

DAI

> Dai is a cryptocurrency that is price stabilized against the value of the U.S. Dollar. Dai is created by the Dai Stablecoin System, a decentralized platform that runs on the Ethereum blockchain.

https://makerdao.com/

EFAMA

European Fund and Asset Management Associaton

http://www.efama.org/SitePages/Home.aspx

ERC

Ethereum Request for Comments. A system similar to IETF's RFC where developers can propose standards. The most known ERC is ERC-20 from Fabian Vogelsteller and Vitalik Buterin.

ERM

Entity relationship model. A technical tool to model and visualize entities and their relations.

EMT

European MiFID II Template

EPT

European Working Group PRIIP Template

ESMA

European Securities and Market Authority

https://www.esma.europa.eu

ETF

Exchange-traded fund. Often used synonymous for a fund that automatically tracks an index.

https://en.wikipedia.org/wiki/Exchange-traded_fund

FCA

Financial Conduct Authority UK https://www.fca.org.uk/

FINMA

Swiss Financial Market Supervisory Authority / Eidgenössische Finanzmarktaufsicht

https://www.finma.ch

https://www.finma.ch/en/authorisation/institutions-and-products-subject-to-the-collective-investment-schemes-act/

ISIN

International Securities Identification Numbers.

> ISINs uniquely identify a security -- its structure is defined in ISO 6166. Securities for which ISINs are issued include bonds, commercial paper, equities and warrants. The ISIN code is a 12-character alpha-numerical code. Think of it as a serial number that does not contain information characterizing financial instruments but rather serves to uniformly identify a security for trading and settlement purposes.

https://www.isin.org/

ISO 4217

Currency codes, usually 3 characters whereas the first two are indicating the country and the 3rd the currency name. Example: CHF, USD, ...

Special currencies start with an X. Therefore, XBT would be the official code for Bitcoin, this isn't yet standardised.

KIID

Key Investor Information Document. A summarized 1-2 pages document that contains most relevant documentation for retail investors.

MiFID II

EU Markets in Financial Instruments Directive. MiFID I originally from 2004 it's successor MiFID II took effect on January 2018.

- https://en.wikipedia.org/wiki/Markets_in_Financial_Instruments_Directive_2004

MOP

Multiple Investment Options

PRIIP

Packaged Retail and Insurance-based Investment Products

> A PRIIP is defined as: an investment where, regardless of its legal form, the amount repayable to the retail investor is subject to fluctuations because of exposure to reference values or to the performance of one or more assets that are not directly purchased by the retail investor; or an insurance-based investment product which offers a maturity or surrender value that is wholly or partially exposed, directly or indirectly, to market fluctuations.
>
> The aim of the PRIIPs Regulation is to encourage efficient EU markets by helping investors to better understand and compare the key features, risk, rewards and costs of different PRIIPs, through access to a short and consumer-friendly Key Information Document (KID). How information in the KID should be calculated and presented is set out in the PRIIPs Regulatory Technical Standards (RTSs).

- Source: https://www.fca.org.uk/firms/priips-disclosure-key-information-documents

Prospectus

> A prospectus, in finance, is a disclosure document that describes a financial security for potential buyers.

https://en.wikipedia.org/wiki/Prospectus_(finance)

Sketch

Digital Mac OS X software for UI/UX Designerss

Solidity

Specialized language to develop smart contracts

SRRI

> The synthetic risk and reward indicator (SRRI) is used to classify investment funds into one of three different risk categories (low risk, medium risk, high risk). It is calculated on the basis of Austrian and European regulatory requirements. This indicator forms an integral part of the Key Investor Information Document (KIID) and gives the historical volatility of the fund unit price on a scale from 1 to 7.

SRRI	Risk category	Volatility intervals
1	Low risk	0% to <0.5%
2		≥0.5% to <2.0%
3	Medium risk	≥2.0% to <5.0%
4		≥5.0% to <10.0%
5		≥10.0% to <15.0%
6		≥15.0% to <25.0%
7	High risk	≥25.0%

http://fundglossary.erste-am.com/srri/

CESR's guidelines on the methodology for the calculation of the synthetic risk and reward indicator in the Key Investor Information Document

UCITS

Undertakings For The Collective Investment Of Transferable Securities

https://www.investopedia.com/terms/u/ucits.asp
https://en.wikipedia.org/wiki/Undertakings_for_Collective_Investment_in_Transferable_Securities_Directive_2009

VaR

Value at Risk

Original project submission

Statement of honesty