Thomas Langlois

thomas.langlois@adacapital.io

Abstract

This document outlines the overall architecture, components, and design choices of the SDK

CardaNoBI – RUST SDK

High-Level SDK Design Document (HLDD)

Contents

[1. Introduction 2](#_Toc161862560)

[2. Overview of the SDK 2](#_Toc161862561)

[3. Architecture 3](#_Toc161862562)

[4. Core Features and Functionalities 5](#_Toc161862563)

[5. Testing and Quality Assurance 7](#_Toc161862564)

[6. Documentation and Support 7](#_Toc161862565)

[Appendices 8](#_Toc161862566)

# 1. Introduction

* **Purpose**: The purpose of the CardanoBI Rust SDK is to commoditize the access to the CardanoBI API for Rust developers by exposing a Rust façade that abstracts the complexities and technicalities of calling the CardanoBI API SaaS REST API endpoints.
* **Scope**: The HLDD will cover the architecture, the core features, the testing strategy, and the documentation.
* **Audience**: All developers interested in using or understanding the CardanoBI Rust SDK.

# 2. Overview of the SDK

* **SDK Components**: The SDK is organized as a traditional Rust package articulated around 3 main pillars:
  + **Utility functions**: where key reusable components (API Client, query parser, constants) are defined.
  + **Domain functions**: where the actual client API endpoints are implemented. These endpoints are organized by domains: Core, BI, Partner.  
    This domain segregation reflects the OpenAPI swagger file and is intended to offer a logical separation between endpoints.
  + **Test functions**: where all unit and functional tests are implemented.
* **Supported Languages and Platforms**: Rust
* **Integration Points**: It is recommended to follow the Rust industry standard approach to manage dependencies and project environment, e.g. to use Cargo the Rust package manager and build system.

# 3. Architecture

* **Design Principles**: The guiding principles are code reusability and modularity.  
  To that end, the code is organized following a Structure-Implementation approach where Structures represent a logic unit for a given domain entity (Epoch, Pool, Block…), their Implementation functions represent logical endpoints. The hierarchical dependency between domain entities is obtained using structures, to hold data and act as a container for a given domain entity, and using method chaining.
* **High-Level Architecture**: The following a high-level architecture diagram shows the SDK components and their interactions.  
    
  A diagram of a software application

  Description automatically generated  
    
  Key points:  
    
  1. The CardanoBI Class is the main entry point to the SDK, once instantiated it is in charge of request an Auth token to the CardanoBI Identity Server to enable secured request/reply exchanges between the SDK and the CardanoBI API SaaS.  
    
  2. Entity Classes are making REST calls to the CardanoBI API endpoints to retrieve the data relevant to the respective endpoint functions exposes by the SDK Entity Classes.

# 4. Core Features and Functionalities

* **API Wrappers**: All API calls are wrapped within the APIClient class which acts as a proxy to the CardanoBI API SaaS instance.  
    
  A screenshot of a computer

  Description automatically generated
* **Authentication and Authorization**: APIClient handles the whole Authenti*cation/Authorization* logic by taking care of exchanging the API Key/Secret for a Bearer Token when the Session with the CardanoBI API SaaS Server is initiated.  
    
  Then upon each REST request, APIClient adds the Bearer Token to the Header of the request to be able to call the targeted endpoints.
* **Error Handling**: APIClient also handles the error management for all client API endpoints by centralizing its logging and its error code generation, which is derived from the error code obtain from either the Backend of the middleware layer.
* **Configuration and Customization**: There are no specific configuration or customization required to use the SDK.  
  The main feature to be aware of is during the instantiation of the CardanoBI, one must provide the API Key, the API Secret and the network.

# 5. Testing and Quality Assurance

* **Testing Strategy**: The SDK will be tested with a series of Unit Tests. We will use the Rust built-in testing framework to perform those tests in a repeatable and consistent manner.
* **Quality Assurance Practices**: In addition to Unit Tests, the SDK will be tested for a performance perspective, where calls to each endpoint will be measured and compared to our nominal response time observed using our baseline Javascript SDK.

# 6. Documentation and Support

* **Documentation**: The documentation for the SDK will be integrated to the existing CardanoBI API Documentation portal by means of Rust code samples embedded with the Documentation.
* **Support Channels**: Email support will be provided to developers requiring further support beyond the CardanoBI API Documentation.

# Appendices

* **Glossary**: Here below we will define important terms and acronyms used throughout this document:
  + **SDK**: Software Development Kit
  + **API**: Application Programming Interface
  + **HLDD**: High-Level Design Document
* **References**: Here below is a list of useful references or external resources related to the SDK:
  + **CardanoBI API Documentation**: <https://docs.cardanobi.io/>
  + **Github repository for the Rust SDK:**  
    <https://github.com/cardanobi/cardanobi-rust>