Core Architecture

The RESTfull API architecture and rules with JSON, as transport data structure as a programming interface.

Application architecture - Clean Architecture based. Does not have to be strictly adhered to but, try to forward Clean Architecture, SOLID, DRY and other best practices.

P.S: SOLID is required everywhere.

Design Patterns:

* Repository Layer - Layer between DB and data consumer
* Service Layer - Core business logic handler for certain model
* Middlewares - For handling auth, secure form CSRF, XSS and etc attacks
* UseCase - If there is some logic, which needs more that 1 model`s service, and handles exactly one process(can be executed by celery)
* Task Queue - Allows handling tasks asynchronously by separating the process of task definition from execution. Implements the Producer-Consumer pattern, where producers enqueue tasks into a message broker, and workers like Celery consume and execute them independently.

Architecture components:

* Celery Worker - Background task manager for parallel, delayed and asynchronous tasks running on another independent process
* Redis - Cache system for fast access to most requested data
* RabbitMQ - Asynchronous message broker for handling task queues and inter-service communication. Ensures reliable message delivery between producers(API, services) and consumers (workers like Celery)
* ElasticSearch - Database, optimized for text operations

**Development style**

Never forget about the coding style standards, as code style cases(pascal, snake…), naming standards and common developers principles, as the same SOLID.

Below given common standards for this project:

* Strict compliance of PEP [8](https://peps.python.org/pep-0008/), [484](https://peps.python.org/pep-0484/) and [257](https://peps.python.org/pep-0257/)
* Maximizing the pursuit python type annotations
* Simple Responsibility - Methods and classes should not be large.

Divide them into smaller objects, and try to reuse them.

* In this application there should not be API versioning, but even if developers will add this functionality, versioning should be by ‘accept’ headers.
* If endpoint returns more that 1 JSON, then endpoint url must end with ‘/’,

else, without it.

Replace {entity} placeholder by entity name on **plural!**

* Endpoints name style:

On base entity crud case:

/api/{entity}/

/api/{entity}/{lookup} (detailed)

For example, user get and update:

GET /api/users/

PUT /api/users/{user\_id}

For more extraordinary case, something like, create a document for user:

POST /api/users/documents/

GET /api/users/documents/

GET /api/users/documents/{document\_id}

What about project management? Do you think? There are some rules, which are generated for better project development and maintenance.

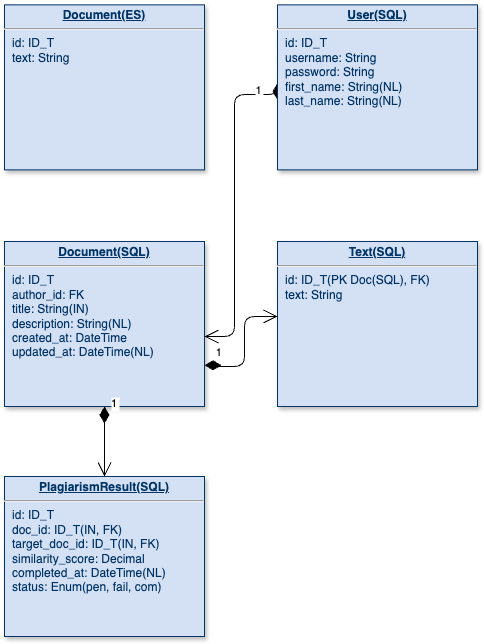
* Env variables - Pre-generate for example file, like .env.example, and set variable names with unfilled values. On CD stage fill unfilled fields by they actual values and then deploy
* Settings - Use pydantic settings for more efficient configurations management
* Simple file, on base case, should not have a lot of code lines. Composite file to folder, by making it a python package(add \_\_init\_\_.py into it) and create more smaller files.
* Folders should have been specified by their common purpose or entity name also. For example with services:

proj\_root/src/services/user\_service.py or proj\_root/src/services/user/…

Here, you can create a whole user folder inside services, and create specified services for users, divided on their own python file.

**Entities**

There are only 5 types of models, which are represented at ER-Diagram for better perceptions and DB development.



**User** - main entity, which is used for authorization and authentication on the system. Password **must** store hashed by SECRET KEY on database.

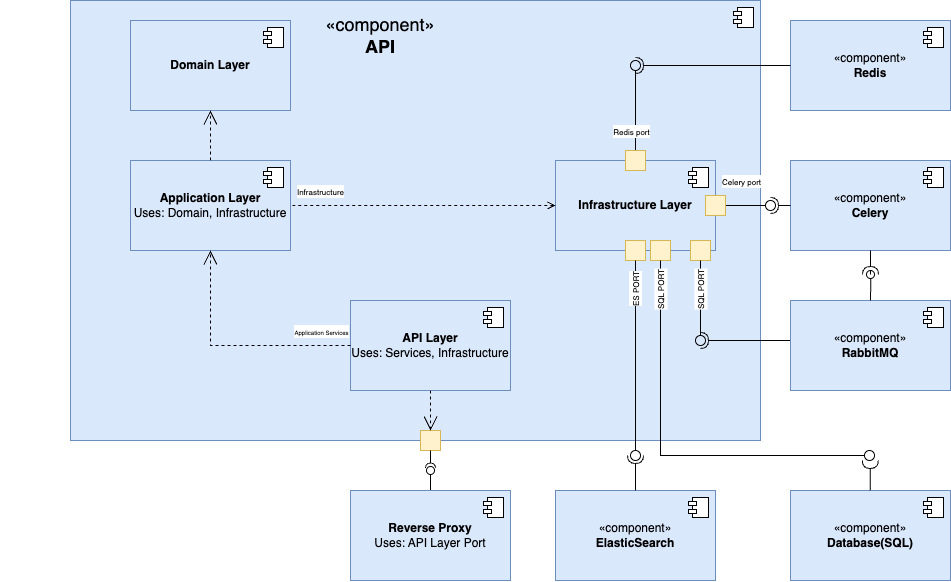
**Document** - In simple words, it’s a text, but without it. Initially, I wanted to store text only in ES DB(ES is used for more complex search аand full-text comparasion) but this tool does not guarantee data integrity. So, it was decided to store a copy of text on SQL db, so it has guaranties for data safety.

**Text -** Primary key is Foreign key to Document. As said in the description on Document model, SQL stores a copy of text, having a primary key of the document, which is great in the current case. And access to text is encapsulated, so if there will be a need to add text versioning, changing this model schema will not be a problem.

**Plagiarism Result -** Result of similarity check.

**System components**

Main components were explained at the start of the documentation , but here is a special detailed diagram, called “Diagram of Components”, which explains how components interact with each other. This diagram was created by using an open source project called “draw.io”. Here is the diagram:



The largest rectangle is the main app “API”, whose architecture is monolith and other small components inside of “API” component, are layers, and whose outside are utils/programs, which are divided from “API” with interfaces.

**Contracts(Endpoints)**

At all, during writing this doc, there are only basic endpoints for methods, like base CRUD with base models fields.

User:

GET /api/users/ - List all users(For development)

GET /api/users/{id} - Get user by identifier

POST api/auth/login - Login here. Returns access & refresh token

POST /api/auth/register - Registration. Will not create this method yet

body:

username: string

password: string

password\_confirmation: string

first\_name: string

last\_name: string

POST /api/users/password/change - Change user's password

body:

password: string

password\_confirmation: string

PATCH /api/users/{id} - Update user info

body:

username: string

first\_name: string

last\_name: string

DELETE /api/users/{id} - delete user

Document:

GET /api/documents/ - Get all documents

GET /api/documents/{id} - Get document by id

POST /api/documents - Create document

body:  
 title: string

description: string

text: string

POST /api/documents/check - Antiplagism check

body:

doc\_id: ID\_T

target\_doc\_id: ID\_T

min\_similarity

PATCH /api/documents/{id} - Update document

body:

title: string

description: string

text: string

DELETE /api/documents/{id} - Delete document