Contents

[Abbreviations 4](#_Toc96463283)

[Figures 5](#_Toc96463284)

[Tables 6](#_Toc96463285)

[1 Introduction 7](#_Toc96463286)

[1.1 Goal and assignments for this thesis 7](#_Toc96463287)

[1.2 What is administration process? 7](#_Toc96463288)

[1.3 What is audit and audit trail? 8](#_Toc96463289)

[1.4 Thesis’s structure 9](#_Toc96463290)

[2 Web API 10](#_Toc96463291)

[2.1 Application Programming Interface (API) 10](#_Toc96463292)

[2.2 Web-based application software and Web API 10](#_Toc96463293)

[2.3 Information system with access via web 11](#_Toc96463294)

[2.4 REST and RESTful API 11](#_Toc96463295)

[2.4.1 REST 11](#_Toc96463296)

[2.4.2 RESTful API 11](#_Toc96463297)

[2.5 RPC 12](#_Toc96463298)

[2.6 GraphQL 13](#_Toc96463299)

[2.7 Other type of web API 13](#_Toc96463300)

[2.8 Conclusion 14](#_Toc96463301)

[3 Technologies stack 15](#_Toc96463302)

[3.1 Types of databases 15](#_Toc96463303)

[3.1.1 Relational database 15](#_Toc96463304)

[3.1.2 Non-relational database 16](#_Toc96463305)

[3.1.3 Choosing database 17](#_Toc96463306)

[3.2 FastAPI + SQLALchemy 18](#_Toc96463307)

[3.2.1 FastAPI 18](#_Toc96463308)

[3.2.2 SQLAlchemy 19](#_Toc96463309)

[3.2.3 Swagger specification and OpenAPI specification 19](#_Toc96463310)

[3.3 React library 19](#_Toc96463311)

[3.4 Docker 20](#_Toc96463312)

[3.4.1 Docker and container 20](#_Toc96463313)

[3.4.2 Networking in Docker and Docker Compose 21](#_Toc96463314)

[4 Problems to solve and implementation 23](#_Toc96463315)

[4.1 Entities in our system 23](#_Toc96463316)

[4.2 Rules for administration process 23](#_Toc96463317)

[4.3 How an instance is transited and received 23](#_Toc96463318)

[4.4 Data model design 23](#_Toc96463319)

[4.5 API design 23](#_Toc96463320)

[4.6 Frontend design 23](#_Toc96463321)

[4.7 Result 23](#_Toc96463322)

[Conclusion 24](#_Toc96463323)

[Source 25](#_Toc96463324)

# Abbreviations

|  |  |
| --- | --- |
| ISO | International Organization for Standardization |
| NIST | National Institute of Standards and Technology |
| CNSSI | Committee on National Security Systems Instruction |
| API | Application Programming Interface |
| URI | Uniform Resource Identifier |
| URL | Uniform Resource Locator |
| HTTP | Hypertext Transfer Protocol |
| HTML | Hypertext Markup Language |
| IS | Information System |
| REST | REpresentational State Transfer |
| CRUD | Create Read Update Delete operations |
| JSON | JavaScript Object Notation |
| XML | eXtensible Markup Language |
| RPC | Remote Procedural Call |
| SQL | Structure Query Language |
| RDBMS | Relational Database Management System |
| ORM | Object Relational Mapper |
| CNM | Container Network Model |

# Figures

[Figure 1 Example of administration process of an aplication for approval of the proposal of the topic of the final thesis 4](#_Toc95945155)

# Tables

# Introduction

## Goal and assignments for this thesis

Our goal is to create an information system (with access via web) with functionality supporting the administration process with elements enabling an audit of the performed tasks. From this goal we must specify which functionality our system must support for administration process, and which performed tasks are audited. The requirements for the system on functionality supporting the administration process are as follows:

* It must be possible to define a sequence of phases and transitions (administration process) by experienced user with proper authorization.
* It must be possible to define who is responsible for every phase in administration process.
* It must be possible to define a structure of form by experienced user with proper authorization.
* Applicant must be able to create an instance from defined form (hereafter referred to as the instance).
* Instance must be able to transit through administration process.

The performed task in administration process is the transition of instance through phases. The creation of process, assignment responsible person for each phase, creation of form is not considered performed task in administration process. It is considered as the creation of administration process. The requirements on audit of the performed tasks are:

* In each phase of administration process, the instance must be recorded what is changed in the instance, who changed it, when it is changed.
* It must be possible to recover instance at any phase.

The assignments of this thesis are:

* Design the information system as a set of cooperating containers (docker compose).
* Study API creation (according to the OpenAPI / Swagger standard) using the FastAPI library.
* Create an information system data model.
* Implement API.
* Prepare technical documentation for API.
* Create a minimalist user interface for the ability to demonstrate functionality.

The necessary output is:

* Source code (on the GITHUB platform) of the container providing the service to authenticated users.
* Documentation for the API (application interface) through which the service is available.
* Source code (on the GITHUB platform) of the container implementing the user interface.

## What is administration process?

In this thesis we can understand an administration process is a collection of phases, where a request of an applicant must traverse through. In each phase responsible person/people will handle the request of the applicant, and then send the request to the next phase, where other person/people will again handle it and send it to the next phase. The process will continue till when the request is completely handled.

Here is an example of an administration process, which is based on Study order of the Dean of the faculty of military technologies, University of Defense in Brno, specified how student propose his / her own topic of the final thesis:

* The student submits a request on the proposal of the topic of the final thesis to the guarantor of the study program (guarantor) through the Vice-Dean for study and pedagogical activities (VDSPA) with the form “Application for approval of the proposal of the topic of the final thesis”.
* The student submits the application for the topic of the final thesis to the head of the department for comment. The head of the department in accordance with the Rector's Measure No. 3/2018 Final Thesis will assess the fulfillment of requirements for the proposed topic of the final thesis, goal, description of the issue and justification of the topicality of the topic to the submitted proposal.
* With the opinion of the head of the department, the student will deliver the application to the Study Group of the Dean's Office of the Faculty.
* VDSPA will request the opinion of the guarantor on all Applications for student’s own final thesis topic. In case of non-approval of the proposed topic or other facts contained in the Application for its own final thesis topic, the guarantor is obliged to state specific reasons for non-approval.

We can visualize the process by diagram below:

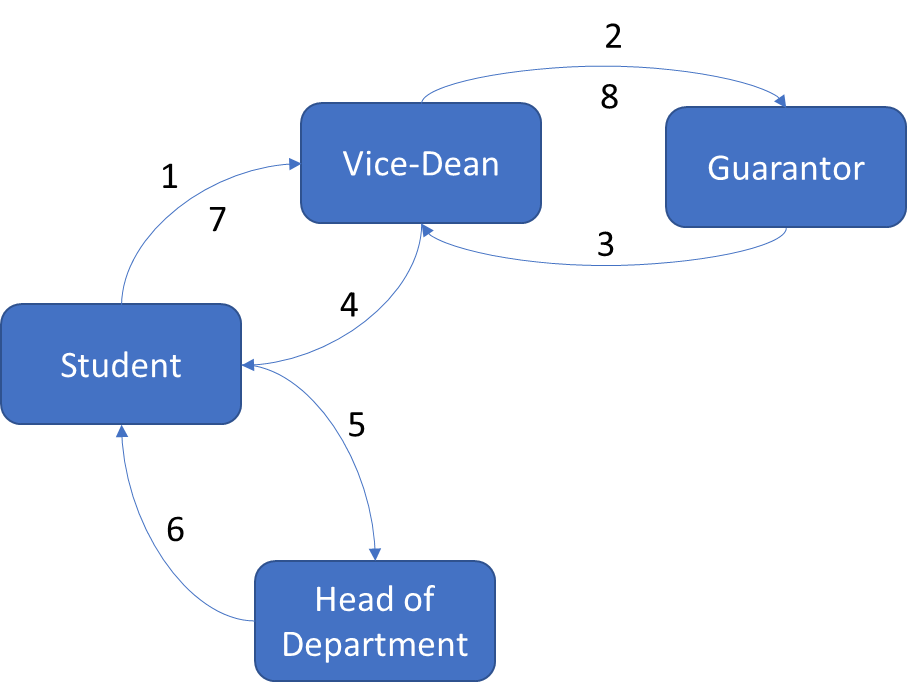


Figure Example of administration process of an aplication for approval of the proposal of the topic of the final thesis

In general, the process will be defined by a regulation and often with a form. Applicant will initialize an instance from this form and send it to the person in next phase. We will discuss about administration process more detail in later chapter.

## What is audit and audit trail?

An audit is often defined as an independent examination, inspection, verification, or review. The subject of auditing may be various in various sector.

The term audit is originally used in financial sector. Anthropologists have found records of auditing activity dating back to early Babylonian times (around 3000 BC). There was also auditing activity in ancient China, Greece, and Rome. The Latin meaning of the word “auditor” was a “hearer or listener” because in Rome auditors heard taxpayers, such as farmers, give their public statements regarding the results of their business and the tax duty due. The practice of modern auditing dates to the beginning of the modern corporation at the dawn of the Industrial Revolution in 18th century with the growth of the joint stock companies and the ownership and management became separate [1]. Audit is defined by Prof. L.R. Dicksee as an “examination of accounting records undertaken with a view to establish whether they correctly and completely reflect the transactions to which they relate”

In information technology sector, the definition of audit doesn’t constrain nor presume the subject to which an audit applies. The International Organization for Standardization (ISO) defined audit in Guidelines for auditing management systems as: “systematic, independent and documented process for obtaining objective evidence and evaluating it objectively to determine the extent to which the audit criteria are fulfilled.” [2] NIST and CNSSI defined audit as: “Independent review and examination of records and activities to assess the adequacy of system controls and ensure compliance with established policies and operational procedures.” [3][4]

NIST and CNSSI also defined audit trail as: “A chronological record that reconstructs and examines the sequence of activities surrounding or leading to a specific operation, procedure, or event in a security relevant transaction from inception to final result.” [3][4]

In case of administration process, we determined the subject of audit is the request of applicant and the audit trail is chronological record of applicant’s request in each phase of administration process, that we can reconstruct and examines the sequence of phases, which the request traversed through in predefined administration process.

## Thesis’s structure

In chapter 2 we will discuss the advantages and disadvantages of 3 types of web API. Then we explain why we choose RESTful API for this thesis.

In chapter 3 we provide.an overview about technologies stack, which are used for development our information system.

In chapter 4 we discuss in more detail about administration process and auditing on performed task, what is the problem we face in development, how we solve it and implement our information system.

The conclusion summarizes the thesis, the benefit and limitation of the thesis and the orientation of our system for development in the future.

# Web API

## Application Programming Interface (API)

According to Bloch J., the idea of application programming interface arose from the idea of subroutine library, when Maurice Wilkes and David Wheeler created programs running in an early computer named EDSAC in the 1940s [6]. The key idea of subroutine and how it is documented is presented in 1952 in Wheeler’s paper *The Use of Sub-routines in Programmes*, in which he said: “…it is usually advantageous to arrange that a programme is comprise of a set of subroutines some of which have been made specially for the particular programme while others are available from a 'library' of standard sub-routines” and “However, even after it has been coded and tested there still remains the considerable task of writing a description so that people not acquainted with the interior coding can nevertheless use it easily. This last task may be the most difficult.”

API is a set of definition and protocol on how two computer programs communicate to each other. API is not the implementation but the specification. In a simple word, a computer program takes an input, process it, and produce an output, then API describe how an input should look like, how to feed the input to the computer program and how the output should look like. For example, A CPU with its instruction set has an API, an operating system with a set of system call has an API, a programming language with a set of standard library functions or built-in functions has an API.

The first key idea of API is the concealment of internal implementation of a computer program, so that it can be reused or combined easily by another program or programmer. The second key idea of API is the independent reimplementation of computer program, so that it does not impact on another program, which is dependent on it, despite of any purpose of the reimplementation, either to upgrade or to degrade it.

A computer program, which implements an API and provides it for another computer program to use, is call API provider. A computer program, which consumes the API, is called API consumer. API provider and consumer can also be referred to programmer, as programmer is the one who write computer program for providing or consuming of another computer program

## Web-based application software and Web API

An application software is a computer program, which solved end-user’s need for a specific task, which can be solved without applying this program. Writing a document can be solved with pen and paper, but it can be solved with word processor. Accounting and auditing can be solved with pen and paper, but it can be solved with an accounting and auditing software. Administration process can be solved with pen and paper, but it can be solved with an application software, occasionally web-based software, which we try to solve in this thesis.

A network-based application software is an application software, which exchanges its data throughout the network. A more specific type of network-based application software is web-based application software (or simply web app), which is created from technologies underlying the World Wide Web (or simply Web) e.g., URI, HTTP, HTML, …

A web app implementing an API is called web API.

## Information system with access via web

An information system (IS) is a set of interrelated elements or components that collect (input), manipulate (process), store, and disseminate (output) data and information and provide a corrective reaction (feedback mechanism) to meet an objective. Processing can be done manually or with computer assistance. A computer-based information system is a single set of hardware, software, databases, telecommunications, people, and procedures that are configured to collect, manipulate, store, and process data into information [7]. An application software is the nearest part of software element to the end user of the information system. Other parts of software element include operating system, networking software, … we do not try to solve in this thesis, because it’s not essential part of the problem.

Our IS must be accessed via web as defined in [1.1](#_What_is_administration), therefore the application software must be a web app. The web app is also required to implement an API and its technical documentation as specified in the assignments of this thesis.

In next sections we examine 3 most popular type or style of designing a web API.

## REST and RESTful API

### REST

Representational State Transfer (REST) is introduced in chapter 5 of the dissertation of Fielding R.T. in 2000 as an architectural style for distributed hypermedia systems [5]. REST is used for guiding the design and development of the architecture for the modern Web. Fielding described REST as a set interaction constraint between components (e.g., client-server, stateless, cache, uniform interface, layered system, code-on-demand), which he calls the style, with the regard to engineering principles (e.g., separation of concerns principle, principle of generality) to derive desired architectural properties of a modern Web architecture (e.g., performance, scalability, simplicity, modifiability, visibility, portability, and reliability).

Fielding R.T also describes the experience and lessons learned from applying REST while authoring the Internet standards for the Hypertext Transfer Protocol (HTTP) and Uniform Resource Identifiers (URI) in chapter 6 of his dissertation. These two specifications define the generic interface used by all component interactions on the Web.

### RESTful API

In the Web development community, a Web Application Programming Interface (Web API or Web Service) conforming to the REST architectural style is referred to REST API or RESTful API.

REST has been developed to represent the model for how the modern Web should work, therefore REST API often refers to the use of URI, HTTP as the standards or technologies used in Web, which is guided by the rationale of REST.

The key abstraction of information in REST is a resource and Uniform Resource Identifier (URI) is used to uniquely identify a resource throughout Web. The term "Uniform Resource Locator" (URL) refers to the subset of URIs that, in addition to identifying a resource, provide a means of locating the resource by describing its primary access mechanism (e.g., its network "location") [8]. The term URL is a more specific term in context of this thesis; thus, we will use term URL instead of URI. A resource of our IS on another hand can be any entity presented in our system e.g., applicant, a form, an instance, a phase, a transition, …

Hypertext Transfer Protocol (HTTP) is one of many communication protocols used in Web to exchange representation of a resource. It has methods e.g., GET, POST, PUT, PATCH, DELETE, … as defined in RFC 2616 [9] to provide interactions (CRUD) with resource identified in URL. These methods are general enough for interaction between API consumer and with almost every type of resource. The exception is the interaction between API consumer and an interaction of a resource with another resource(s), which requires a specific parameter(s) than states of resource itself. For example, the transiting of an instance to next phase requires information about responsible person/people, which is not the state of the instance itself. Searching throughout a collection of resource also a typical problem with REST API. Although we can consider an interaction between resources like another resource as definition about resource in [8]: “… abstract concepts can be resources, such as the operators and operands of a mathematical equation …”, we will mix the concept about resource and interactions and cause the confusion and ambiguity for the programmers, as maintainer or consumer.

A data format of representation of a resource in REST API is often, but not always, JSON or XML. In this thesis we use JSON as data format of representation of resource because of its more simplicity in syntax and less overhead than XML.

The reason for applying REST in API design of our system as Fielding said: “… when applied as a whole, emphasizes scalability of component interactions, generality of interfaces, independent deployment of components, and intermediary components to reduce interaction latency, enforce security, and encapsulate legacy systems” [7].

Pros:

* Easy to implement, maintain and scale up
* Suitable for CRUD resource and showing relationship among resources

Cons:

* Not suitable for specific action of a resource
* Big payload in response
* Multiple HTTP round trips is required when we want to retrieve representation of multiple resources.

## RPC

Remote Procedural Call (RPC) In distributed systems is when a client program calls a function whose implementation is in a remote server. In Web API, RPC-style APIs focus on actions. Each endpoint represents an action the client can perform on the server. The action is a function that is called on the server. Like a normal function that receives parameters and returns a value that is sent as a response to the client. For an RPC API that uses HTTP as the transport protocol, the method or function is places in the URL and the arguments are placed either in the query string or body.

RPC style is suitable for APIs that expose a variety of actions of resources, which might have more nuances and complications than can be encapsulated with CRUD. RPC partially solved the problem of REST, but it introduces a new problem. Every type of resource will have some common action and some its own specific action. Firstly, we repeatedly implement the common actions on every type of resource. Secondly, it’s not ideal to standardize, manage, maintain, or consume with such diverse list of actions, cause that list will expand over time when we want to add more action or more type of resource with its own specific actions.

Pros:

* Suitable for specific action on a resource

Cons:

* Only apply for specific solution, lack of standardization
* Can lead to function explosion if more action is added

## GraphQL

GraphQL is a query language for APIs, which was developed internally by Facebook in 2012 and was publicly released in 2015 [10] and has been adopted by API providers like GitHub, Yelp, and Pinterest.

GraphQL allows consumer to define the structure of the data required, and the server returns data with exactly that structure, thus reduce the redundant data in response. GraphQL has a single endpoint to interact with and two type of operations, query for read-only fetch data and mutation to write followed by a fetch.

GraphQL reduce the redundant data in payload of a response and number of HTTP roundtrips in the comparison with REST API. It is suitable for resources with a large amount of data field in its state and for the resource in high hierarchical relationship with other resources.

The drawbacks of GraphQL are the complexity in the implementation, additional processing to parse queries and verify parameters. Therefore, GraphQL is not suitable for a simple API. Nevertheless, GraphQL can be used in combination with REST API, and it can be considered when system become larger. For simplicity of our system in implementation, we do not use GraphQL in this thesis, but it is considered a potential solution when our system evolves and becomes more complex.

Pros:

* Less redundant data in payload of response
* Less HTTP roundtrip

Cons:

* Additional complexity
* Additional processing
* Not suitable for simple API

## Other type of web API

Other type of web API include SOAP API or event-driven web API e.g., Webhook, WebSocket, HTTP Streaming, which are either no longer in use or server-to-server communication or bidirectional client-server real-time communication or sending an infinite chunk of data in infinite numbers of response, are considered unnecessary for the requirement of our IS, so we won’t make any further discussion about them in this thesis. Nevertheless, it is possible to combine these type of web API with other types of web API if more requirements on our IS are made.

## Conclusion

In this chapter, we discuss about what is API, web API, IS with access via web and three types of web API, which are mostly used in web development. We evaluate pros and cons of each type of web API, and how they affect the decision to use REST in the design of our web API.

In the next chapter we will examine technologies stack used for development our IS in this thesis.

# Technology stack

In this chapter we will consider which technologies will be used in the development for our IS and why we choose them. Because there is no requirement on which database will be used, firstly we examine types of databases and I as author of this thesis will choose the type of database according to the requirements of the IS, the possibility to implement, my own ability and experience with database. Second assignment is to create API using FastAPI library, hence we will introduce the FastAPI and additional tool using with it. Next, we will introduce React as technology for creation a minimalist user interface for the ability to demonstrate functionality as being assigned in chapter 1. We choose to introduce Docker in the last section, because we think Docker only has meaning when other software preexists, even when the first assignment for this thesis is about Docker.

## Types of databases

Two requirements we must consider when we choose a type of database is the possibility to define administration process and the possibility to define a structure of form. An administration process is a sequence of phases and transitions, or nodes and edges in term of graph theory. A form is an entity in our system, and it contains a varying number of fields in it. These fields are the structure of the form, and each form will have its own structure. Our system must be able to store such administration process and such arbitrary structures of different forms. Users must be able to initialize an administration process and instantiate an instance from corresponding form and fill data to it. Our system must be able to store current phase of the instance and data filled by user. We will consider two types of modern databases, which can fulfill these requirements, and those are relational and non-relational database, also known as SQL or NoSQL database. Then we will choose one type of database and its specific instance to use in our system.

### Relational database

Relational database was invented by E. F. Codd at IBM in 1970 in his research paper *A Relational Model of Data for Large Shared Data Banks*. A relational database stores data in relations, which the user perceives as tables. Each relation is composed of tuples, or records, and attributes, or fields. The physical order of the records or fields in a table is completely immaterial, and each record in the table is identified by a field that contains a unique value (also known as primary key). The relational model categorizes relationships as one-to-one, one-to-many, and many-to-many. A relationship between a pair of tables is established implicitly through matching values of a shared field (also known as foreign key). Tables and their relationships form a schema of a database. The schema is predefined and is not often changed, because change in schema required redefining database and data migration. Therefore, relational database is not ideal for unstructured data, and it slow down the development process because database schema usually changes overtime.

The most common way of interacting with relational database systems is using Structured Query Language (SQL), which allows for specific queries to be built that help build, search, and filter data across one or multiple tables.

A relational database management system (RDBMS) is an application software, which provide an API and possibly a user interface and allows us to create, maintain, modify, and manipulate a relational database.

The most popular RDBMS are Oracle, MySQL, Microsoft SQL Server, PostgreSQL, SQLite, MariaDB, …

### Non-relational database

A non-relational database, sometimes called NoSQL (Not Only SQL), is any kind of database that doesn’t use the tables, fields, and columns structured data concept from relational databases.

There are four types of NoSQL Databases [12]:

#### Document Databases

Document databases store data in documents, which are usually JSON-like structures that support a variety of data types. These types include strings; numbers like int, float, and long; dates; objects; arrays; and even nested documents. The data in document is stored in key/value pairs.

A collection is a group of documents, which usually store documents that have similar contents. Not all documents in a collection are required to have the same fields, because document databases have a flexible schema.

Retrieving data from document database vary between document databases and often is proprietary. For example, to get all documents from collection users:

In MongoDB query takes the form of method chaining and data can be queried:

db.users.find({})

ArangoDB use its own query language and data can be queried:

FOR doc IN users

RETURN doc

These are equivalent in SQL database as:

SELECT \* FROM users

Some of the most well-known document databases or document database services are MongoDB, ArangoDB, Amazon DynamoDB, Google Cloud Firestore, …

#### Key-Value Stores

Key-value databases (also known as key-value store) stores data in a “key-value” format and optimized for reading and writing data. The data is identified by a unique key or several unique keys. For each key, there is exactly one value, which can be simple data types like strings and numbers or complex objects.

This type of database implements a hash table data structure (also known as dictionary, map, hash map, symbol table), that is the key is hashed to a numerical value (the hash) and the value is stored in position defined by the hash. The value is retrieved by the key in constant time.

Key-value stores have no query language. Values cannot be queried or searched upon. Only the key can be queried. Data is written (inserted, updated, and deleted) and queried based on the key.

Redis is the typical examples of key-value store.

#### Column-Oriented Databases

Column-Oriented databases store data by column rather than by row as relational database.

In relational database each row contains all columns related to all attributes of an entity instance. The table is a sequence of rows, and each row is stored as a block in disk. A row as a block is fetched with all its columns (instance’s attribute). The problem happens if we want to retrieve a column or a small groups column of all rows, because all block of table must be fetched. Therefore, it is more efficient to read data in such fashion if we store data related to an instance per column in table.

In column-oriented database each row contains data of an attribute related to all instances and is stored as a block in disk. An attribute or set of attributes of all instances is retrieved by fetching one block or a set of blocks without fetching whole table. To effectively query data in column-oriented databases, each unit of data in a block is a set of identifier/value pairs. Column-oriented database is fast in aggregate operation for example average age of user. Adding attribute is also fast by adding just adding another block on disk. Adding instance is slow because each attribute of all instances is continuously store in disk.

SQL can be used as query language in column-oriented database.

Some examples of column-oriented databases are Apache Cassandra, HBase.

#### Graph Databases

Graph databases solve the limitation of relational databases in schema by treating relation as data. The data of entity instance is stored in nodes and the relation is store in edges. Each node is categorized by adding label to it, as an entity instance is categorized in a table of relational database. Node stores attributes (properties) in form of key/value pair. Edge also has label, possibly attributes (properties), and additionally direction. The relation is added by simply adding a new edge without redefining database and data migration.

Graph database is suitable for dealing with highly interconnected entities and is optimized to capture and search the connections between data elements, overcoming the overhead associated with JOINing multiple tables in SQL.

Graph database like Neo4j, which is one of the most well-known graph databases, use Cypher Query Language to retrieve data from graph. It was inspired by SQL and has similarity with SQL.

### Choosing database

Consider the first and the third requirement on our IS, those are possibility to define phases and transitions of an administration process and possibility to define form structure for such process.

The phases and transitions are more like a nodes and edges in graph theory. Each phase in process can point to another phase in the same process, and the transition is created for store that relation. For different administration process it must be different phases and transitions. To not be confused with the graph database, graph database is use for highly interconnected entities and the relations between entities are prone to change. Phases and transitions are only two entities in our system and the relationship between them are not prone to change. We can easily store phases and transitions in two tables as the example shown below. With these two tables we can easily describe any administration process, including add, modify, or delete transition between two phases. There is no need to use a graph database for fulfilling this requirement. We will extent this topic in relation between these two table in chapter [4](#_Problems_to_solve).

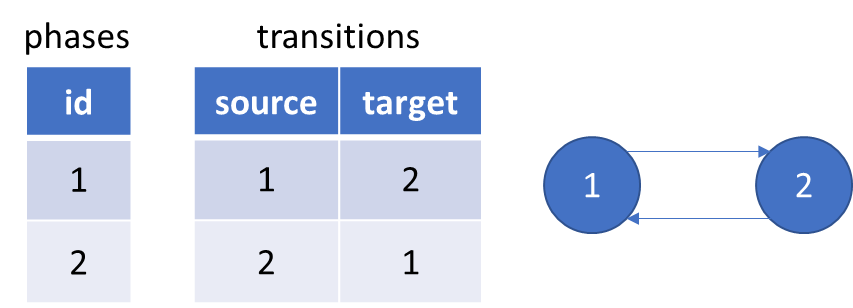


Figure Relational database to store phases and transitions

The form is the structure for the instance. The structure of the form defines that, in the instance how many fields are presented and can be filled by user. The instance can be instantiated from the form and the user can fills data to it. In the programmer’s point of view a form is like a class and an instance is like an object, instantiated from the class. The problem is that the structure of the form is not predefined by programmer like a table in database to store data filled by user, because this structure itself is the data and is defined by user, and we must store it. Then another user like an applicant wants to instantiate an instance and fill data to this instance, we must store this data in a manner, that we can retrieve this data and know which field it belongs to.

One possible solution is that we store this structure of form in a document database, in which a collection name “forms” will store all form’s structure in documents with JSON-like format. These documents can have different fields because of the nature of flexible schema of document database. The key can be the name of the field and the value can be any metadata describing this field in any data format. Instances will be stored in documents of another collection name “instances”, in which field’s name can be derived from one document of the collection “form” as a key and field’s value is data from user, metadata of the field can be used to validate this value.

But also, as being said we can split the structure of form and the data filled by user and store them separately in different tables as normal data. This solution can be achieved by using relational database. We will discuss about this solution in the chapter [4](#_Problems_to_solve).

The reason for choosing relational database in the development our information system is learning something entirely new, including design method and tool, is a barrier to adoption. It takes time to master a tool with proper design method. Therefore, with time limitation in implementing our information system, I as the author of this thesis decided to choose relational database to develop our IS. The RDBMS I choose to use in this thesis is PostgreSQL.

## FastAPI

### FastAPI

As introduced in the documentation of [FastAPI](https://fastapi.tiangolo.com/), it is a modern, fast (high-performance), web framework for building APIs with Python 3.6+ based on standard Python type hints.

FastAPI is based on and fully compatible with OpenAPI. It automatically generates a specification for all defined API conformed with the OpenAPI standard. The specification includes API paths, the parameters they take, etc. The file for this specification is in JSON data format (also known as JSON schema) with name openapi.json. This file is used for interactive documentation system like Swagger UI to visualize and interact with the API’s resources.

Graphical user interface, text, application

Description automatically generated

Figure The interactive documentation system auto created by SwaggerUI from openapi.json

### SQLAlchemy

The language used to implement web API is Python. Python most well-known database drivers for PostgreSQL is psycopg2. But to simplify the development process we use SQLAlchemy, a Python SQL toolkit and Object Relational Mapper (ORM).

### Swagger specification and OpenAPI specification

The OpenAPI Specification was originally based on the Swagger Specification (version 1.0 to version 2.0), donated by SmartBear Software. In 2015 Linux Foundation announced the Open API Initiative, which extend the Swagger 2.0 specification and rebranded it as OpenAPI specification [11].

Swagger specification and OpenAPI specification are used to describe and document RESTful APIs. An OpenAPI definition can then be used by documentation generation tools to display the API, code generation tools to generate servers and clients in various programming languages, testing tools, and many other use cases.

An example of documentation generation tool is Swagger UI, which is used in FastAPI.

## React

Another assignment for this thesis is to create a minimalist user interface for the ability to demonstrate functionality. To complete this assignment, we use ReactJS and CytoscapeJS

React is a JavaScript (JS) library for building user interfaces. It can be used to create a single-page web app, which allows user to request a web page (HTML, JS, CSS, …) only once and this page using JS code will request the data from API and update the DOM tree. It creates a virtual DOM used for comparing and updating the DOM tree, which will be rendered to the browser. The process of comparing and updating the DOM tree is called reconciliation. React implements a heuristic diffing algorithm in reconciliation based on two assumptions:

* Two elements of different types will produce different trees.
* Child elements may be stable across different renders with a key prop.

These two assumptions help React reduce the time complexity from O(n3) to O(n) in a comparison with generic solutions to generate the minimum number of operations to transform one tree into another [13].

Another consideration when building a user interface for our system is the visualization of an administration process for user, which we will implement by using CytoscapeJS.

## Docker

The first assignment of this thesis is to design the information system as a set of cooperating containers (docker compose).

### Docker and container

Docker is an OS-level virtualization software (often refer to containerization), which packages an application and its dependencies into a software unit, called container. It reduces the waste of resource i.e., CPU, RAM, storage, OS license for each application software with virtualization technology by using shared host OS for all containers run on it. It also provides a way for quicker build, test and deploy without platform compatibility barrier [13][14].

Chart, treemap chart

Description automatically generatedGraphical user interface, application

Description automatically generated

Figure Containerization vs Virtualization.

Source: <https://www.docker.com/resources/what-container>

A container is a standard unit of software that packages up code and all its dependencies, so the application runs quickly and reliably from one computing environment to another. A docker container images (shortly image) is a lightweight, standalone, executable package of software that includes code, runtime, system tools, system libraries and settings, which are required to run an application. Images become containers when they run on Docker Engine at runtime.

### Networking in Docker and Docker Compose

Docker network design is based on Container Network Model (CNM), which contains 3 main components: Sandbox, Endpoint and Network [13]

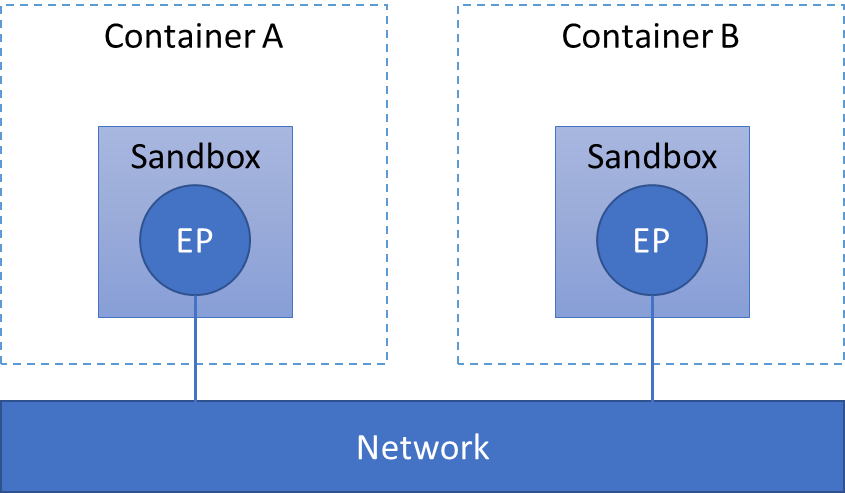


Figure Docker network design based on CNM

* A Sandbox contains the configuration of a container's network stack. This includes management of the container's interfaces, routing table and DNS settings. A Sandbox may contain many endpoints from multiple networks.
* An Endpoint connect a Sandbox to a Network as a virtual network interface (e.g., veth). An Endpoint can belong to only one network, and it can belong to only one Sandbox, if connected.
* A Network is a software implementation of a switch, which group Endpoints together and enable them to communicate with each-other directly.

CNM is implemented by libnetwork and driver. The libnetwork provide an API to for user like Docker or third party to create and manage CNM objects for example Network or Endpoint object. The actual implementation of creation and manage these objects by user like Docker or third party is called driver. Several drivers exist by default in Docker, and provide core networking functionality e.g., bridge, host, overlay, ipvlan, macvlan. Other network driver plugins are provided by third-party in Docker Hub.

When we start Docker, a default bridge network is created automatically, and newly started containers connect to it. Containers running on the same host can communicate with each other in a default bridge network by using only IP addresses. A user-defined bridges provide automatic DNS resolution between containers.

Docker Compose (shortly Compose) is a tool for defining and running multi-container Docker applications. By default, Compose sets up a single network for our app. Each container joins the default network and is reachable by other containers on that network by container name defined in Compose file (usually in YAML format).

Docker can also allow a port on container to be mapped to a port on host. Any traffic with configured port come to the host will be directed to the container.

Our information system is a set of cooperating containers using Docker Compose as first assignment for this thesis. We will use Docker and Docker Compose in development of our IS. The structure of cooperating containers will be presented in the next chapter, after we present all other services.

## Summary

In this chapter, we introduced 2 types of databases, those are relational and non-relational database and I as the author of this thesis decided to choose PostgreSQL as database management system to store data. We also introduced other framework and library to develop our IS, those are FastAPI, React and Docker. These framework and library is chosen in consideration with the requirements on our IS. The next chapter will present the design of our IS and how we implement our IS using these framework and library.

# Problems to solve and implementation

## Entities in our system

## Rules for administration process

## How an instance is transited and received

## Data model design

## API design

## Frontend design

## Result

# Conclusion

# Source

1. Hayes R., Dassen R., Schilder A., Philip W., *PRINCIPLES OF AUDITING, An Introduction to International Standards on Auditing.* Second edition. 2005. ISBN 0 273 68410 8
2. International Organization for Standardization, *ISO 19011*, *Guidelines for auditing management systems.* Third edition. 2018.
3. NIST *Security and Privacy Controls for Information Systems and Organizations.* <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-53r5.pdf>
4. CNSS 4009. *Committee on National Security Systems (CNSS) Glossary.* <https://www.cnss.gov/CNSS/openDoc.cfm?X9siqniuE4GU0To+emCGBA==>
5. Fielding R.T., *Architectural Styles and the Design of Network-based Software Architectures.* 2000. <https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>
6. Bloch J., *A Brief, Opinionated History of the API.* 2018. <https://www.infoq.com/presentations/history-api/>
7. Stair R. M., Reynolds G. W., Fundamentals of Information Systems. 6th edition. 2012. Course Technology. ISBN-13: 978-0-8400-6218-5
8. Berbers-Lee T., et al., *Uniform Resource Identifier (URI): Generic Syntax.* 2005. <https://datatracker.ietf.org/doc/html/rfc3986>
9. Fielding R., et al., *Hypertext Transfer Protocol -- HTTP/1.1.* 1999. <https://datatracker.ietf.org/doc/html/rfc2616>
10. Facebook, Inc., *GraphQL.* 2015. <https://spec.graphql.org/July2015/>
11. GitHub repository of OpenAPI specification <https://github.com/OAI/OpenAPI-Specification/>
12. MogoDB, *Understanding the Different Types of NoSQL Databases.* <https://www.mongodb.com/scale/types-of-nosql-databases>
13. Docker, *Use containers to Build, Share and Run your applications.* <https://www.docker.com/resources/what-container>
14. Poulton N., *Docker Deep Dive*, *Zero to Docker in a single book. 2020 edition.*
15. React, *Reconciliation.* <https://reactjs.org/docs/reconciliation.html>