

Project Specification: KYC

Client: Neoloan AG

Contractor: NeoTheOne

Contact

Lea Pellikan <lea.pellikan@gmail.com>

Felix Wortmann <felix.wortmann@stud.tu-darmstadt.de>

Leon Chemnitz <leon.chemnitz@stud.tu-darmstadt.de>

Chen Yang <chen.yang@stud.tu-darmstadt.de>

Lei Zhuang <lei.zhuang@stud.tu-darmstadt.de>

Version	Date	State	Changes
1	2.12.2022	Work in Progress	<ul style="list-style-type: none">• Document initialization
2	5.12.2022	Work in Progress	<ul style="list-style-type: none">• Rework Deliverables
3	6.12.2022	Ready for first feedback	<ul style="list-style-type: none">• Added first version of requirements• Added Deployment environment• Reworked Risk section
4	14.12.2022	Work in Progress	<ul style="list-style-type: none">• Legal Regulations
5	15.12.2022	Work in Progress	<ul style="list-style-type: none">• Completed user stories
6	15.12.2022	Work in Progress	<ul style="list-style-type: none">• Work in feedback
7	16.12.2022	Ready for submission	<ul style="list-style-type: none">• Final touches
8	20.12.2022	Ready for signature	<ul style="list-style-type: none">• changed spelling of Neoloan

Table of contents

- 1. Project Description
 - 1.1 Project Introduction
 - 1.2 Customer Vision
 - 1.3 Existing State
 - 1.4 Desired State
 - 1.4.1 KYC-Wizard
 - 1.4.2 Dashboard
 - 1.4.3 APIs for integration
 - 1.4.4 Multi-Tenancy
 - 1.5 Basic Terms in the Project Environment
- 2. Communication Agreements
 - 2.1 Processes and Procedures
 - 2.2 Handling Change Requests
- 3. Requirements
 - 3.1 Definition of Done
 - 3.2 User Roles
 - 3.3 Epics
 - 3.4 User Interface
 - 3.5 API Exposure
 - 3.6 Non-Functional Requirements / Environment requirements
 - 3.7 Acceptance Criteria
- 4. Environments
 - 4.1 Deployment Environment
 - 4.1.1 Containerization
 - 4.1.2 Container Orchestration
 - 4.1.3 Cluster Virtualization
 - 4.1.3 Versions
 - 4.2 Environment instances
 - 4.2.1 Development Environment
 - 4.2.2 Production Environment
 - 4.2.3 Testing Environment
 - 4.3 Tools
- 5. Deliverables
 - 5.1 Source Code
 - 5.2 Documentation

5.3 Kubernetes / Docker

6. Risk Mitigation

6.1 Risk Matrix

6.2 Risks and Risk Avoidance

6.2.1 Absence of a Responsible Person

6.2.2 Illness or Vacation

6.2.3 Communication Problem

6.2.4 External Delays

6.2.5 Failed External API

6.2.6 Unforeseen Bugs

6.2.7 Knowledge Gap in the Team

7 Legal Regulations

7.1 Vertragspartner

7.1.1 Auftraggeber

7.1.2 Auftragnehmer

7.2 Vertragsgegenstand

7.3 Nutzungs- und Urheberrechte

7.4 Schadensersatz- und Wartungsansprüche

7.4.2 Gewährleistung und Wartung

7.4.2 Schadensersatzansprüche

7.5 Vertraulichkeitsvereinbarung

7.6 Anwendbares Recht und Gerichtsstand

7.7 Abnahme

1. Project Description

The aim of this project is the development of a software product commissioned by Neoloan AG which will be referred to as the *client*. The development is executed by the student group NeoTheOne which will be referred to as the *team*.

This section presents the product and its desired state at the end of the project. In addition, basic terms in the project environment are defined in [Section 1.6](#).

1.1 Project Introduction

The core of the project is the development of a Know Your Customer (KYC) application. KYC is a process for identifying and verifying a natural or legal person based on legal requirements before a syndicated loan is handed out to them by a consortium of banks. This is necessary for the prevention of money laundering and is mandated by the German Anti-Money Laundering Act and the EU Anti-Money Laundering Directive. KYC is used by lenders to assess creditworthiness. Before entering into a business relationship, credit institutions must carry out an examination of the borrower to minimize risks during business activities.

1.2 Customer Vision

The core business of the client is to provide a platform for its customers which are mostly banks to hand out and manage syndicated loans. Customers have to comply with many legal regulations when handing out a syndicated loan and the client wants to ease the process of ensuring this compliance.

In order to achieve this goal, the client proposed to build a web application that guides their customers through the KYC process in a wizard-like form. Additionally, the application should display all the information aggregated in the KYC process like legal ownership of companies and individuals, etc. in a graph.

The client does not expect a fully developed product at the end of this project with all details of the KYC process realized, but rather wishes to see continuous progress throughout the project. After the end of this project, the client plans to test and further develop the application in order to integrate it into their main product.

1.3 Existing State

At the time of starting this project, the development of the application has not begun and there is no existing application or code base. The client aims to provide the team with pre-made components for the frontend of the application but backend and database have not been initialized. Furthermore, the client aims to provide access to application programming interfaces (APIs) that allow the querying of data necessary for the development of the application.

Additionally, the client has provided access to tooling used for the development of the project, details of which can be found in [Section 4.3](#).

1.4 Desired State

After the development of the application, the following main functions should be implemented:

1.4.1 KYC-Wizard

A user of the application should be guided through the KYC process in a wizard-like form. The application should prompt the user to enter the required information in forms. These forms should be prefilled with data accessed through the APIs, provided by the client.

1.4.2 Dashboard

A user of the application should be able to see the aggregated information on a dashboard. This information should be structured in a graph.

1.4.3 APIs for integration

The application should expose an API so that its functionality can be integrated into the existing microservice infrastructure of the client.

1.4.4 Multi-Tenancy

The application should provide multi-tenant access to its users.

1.5 Basic Terms in the Project Environment

Loan: In finance, a loan is the lending of money by one or more individuals, organizations, or other entities to other individuals, organizations, or other entities.

Syndicated loan: A syndicated loan is a substantial loan provided to a borrower requiring a large amount of money (1 million € or more) by several lenders in a consortium.

Multi-Tenancy: Multi-Tenancy is an architecture where multiple tenants share the same physical instance of an application.

Legal entity: An entity refers to a person or organization possessing separate and distinct legal rights, such as an individual, partnership, or corporation.

Natural entity: In jurisprudence, a natural person is a person that is an individual human being, distinguished from the broader category of a legal person, which may be a private or public organization.

GwG (German Anti-Money Laundering Act): Act on the detection of proceeds from serious crimes.

2. Communication Agreements

This chapter describes how the project is executed, i.e. the processes by which the team operates and how the client and the team interact. The technologies used for organization and communication are also shown.

2.1 Processes and Procedures

A modified version of Scrum is used to implement the project, the details of which are described in the next paragraph. The client has permanent access to all resources of the project except for a team internal communication tool and can always access the latest deployable state of the application via a running instance, as soon as this is feasible.

A representative of the client can be reached via Slack. Project planning and task management are conducted via Jira.

Similar to ordinary scrum, project development is executed in sprints. The sprint duration is 2 weeks but this can be adjusted if deemed necessary by the client or the team if both parties agree.

Sprints start with the planning phase. The team has created tasks from defined user stories, a subset of which will be chosen for implementation in the coming sprint and organized in a sensible priority ranking, based on the capacities of the team.

In a meeting with all parties at the beginning of a sprint, this sprint plan will be presented and discussed with the client. Changes are made if necessary adjustments are revealed in the discussion.

In the same meeting, a review of the previous sprint is conducted with the client.

Successes and problems in the implementation of tasks in the last sprint are presented by the team and discussed with the client. Furthermore, the team presents an analysis of their remaining working time budget in relation to implemented and remaining tasks. If necessary, this information is presented in graphical form as a burndown chart. The planning for the new sprint is adjusted accordingly if deemed necessary.

User stories that are considered implemented by the team are assessed by the client as part of the review. Details of this process can be found in [Section 3.1](#).

At the end of the meeting, the team and the client have an open discussion about the work process and team dynamics in general. This part of the meeting is equivalent to the retrospective of the ordinary scrum process.

2.2 Handling Change Requests

The client can propose changes at any time during the project. The team considers how much work this change is likely to entail and discusses with the client how this change is going to affect the current planning and task prioritization of the project. The client has the opportunity to reconsider if they want the change to be integrated. The integration of a change into the project planning can only happen at the end of a sprint, during the planning phase of the next sprint.

3. Requirements

The application that is to be developed by the team in the context of this project is subject to many legal regulations. A full implementation that satisfies all legal constraints is unlikely to be achieved in the scope of this project. Because the client is aware of this, the project is executed in an agile style with adapting functional and non-functional requirements.

To specify the product that is going to be developed in this project in the best way currently possible, user stories have been defined. All user stories that have been identified up to this point with the client are listed in [Section 3.3](#) a graphical representation can be found in Figure 3.3.1 to give an overview. All user stories are organized in Epics and are prioritized in order of occurrence in [Section 3.3](#). This prioritization is subject to change and can be adapted in the sprint planning phase if necessary. The user stories are the functional requirements but a non-fulfillment of a User Story by the end of the project does not mean that the project is considered a failure or incomplete. Instead, the team attempts to implement as many user stories as reasonably possible in the given timeframe. To determine whether a user story can be considered complete and fully implemented, a definition of done is specified in [Section 3.1](#). The user roles that are expected to operate the application are defined in [Section 3.2](#) and a description of the current vision of the user interface is presented in [Section 3.4](#). Furthermore, a description of the application programming interface (API) the application is to expose is presented in [Section 3.5](#), and a description of all non-functional requirements is in [Section 3.6](#). Finally, [Section 3.7](#) describes the acceptance criteria that need to be met so that all requirements can be considered fulfilled.

3.1 Definition of Done

For a user story to be considered done, it has to pass certain acceptance criteria. In addition to the functionality that is implemented for a given user story, a test suite needs to be developed in the style of critical path testing. That is, a positive test needs to be implemented that asserts the correct behavior of a functionality under standard or expected user input, and a negative test needs to be implemented that asserts the correct behavior under non-standard or incorrect user input.

Furthermore, the code that is developed in the implementation of a user story should adhere to the style described in the book “Clean Code, a handbook of agile software craftsmanship” by Robert C. Martin¹. The assessment of whether the implementation of a user story is accepted and whether the user story can be considered done is conducted by a representative of the client in the sprint review phase at the end of each sprint. A user story fails to be accepted if the representative identifies defects. The correction of these defects is scheduled for the next sprint, after which the user story is reassessed.

3.2 User Roles

At the moment only one user role is intended for the application. The role is that of a customer of Neoloan i.e. the representative of a bank that wants to perform the KYC process before handing out a syndicated loan to one of its clients.

A user with this role will be referred to as *bank user* and the legal entity that wants to take out a syndicated loan from a bank user will be referred to as *bank client*.

¹ Martin, R. C. (2009). *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall.

3.3 Epics

1. Identification of all parties associated with the bank client.

User Stories:

- 1.1. As a bank user, I want to view the current hardcopy printout of a bank client, so that I can identify the legal entity.
 - 1.2. As a bank user, I want to download the current hardcopy printout of a bank client, so that § 12 Abs. 2 GwG is fulfilled.
 - 1.3. As a bank user, I want to fill in the information for a natural person in a form, so that § 11 Abs. 4 GwG (1) is fulfilled.
 - 1.4. As a bank user, I want to fill in the information for a legal person in a form, so that § 11 Abs. 4 GwG (2) is fulfilled.
 - 1.5. As a bank user, I want to view all related current hardcopy printouts of a bank client, so that I can identify the structure of the corresponding legal entity.
 - 1.6. As a bank user, if a representative body acts on behalf of a client I want to confirm that the representative body is authorized to do so so that § 10 Abs. 1 GwG (1) is fulfilled.
2. Determination and identification of all beneficial owners of a bank client.

User Stories:

- 2.1. As a bank user, I want to view a list or graph of all beneficial owners of a bank client, so that § 10 Abs. 1 GwG (2) is fulfilled.
- 2.2. As a bank user, I want to fill in the information of a beneficial owner of a bank client, so that § 11 Abs. 5 GwG is fulfilled.

3. Determination of the responsibility, ownership, and control structure of a bank client.

User Stories:

- 3.1. As a bank user, if the client that wants to take out a loan is a legal entity, I want to view the responsibility structure of a bank client as a graph, so that § 10 Abs. 1 GwG (2) is fulfilled.
 - 3.2. As a bank user, if the client that wants to take out a loan is a legal entity, I want to view the ownership structure of a bank client as a graph, so that § 10 Abs. 1 GwG (2) is fulfilled.
 - 3.3. As a bank user, if the client that wants to take out a loan is a legal entity, I want to view the control structure of a bank client as a graph, so that § 10 Abs. 1 GwG (2) is fulfilled.
4. Examination of risk factors and PeP-state of beneficial owners of a bank client as well as risk assessment.

User Stories:

- 4.1. As a bank user, I want to fill in the purpose of a loan of a client in a form, so that § 10 Abs. 1 GwG (3) and § 10 Abs. 2 GwG is fulfilled.
 - 4.2. As a bank user, I want to fill in the amount of deposited asset values or transactions of a bank client, so that § 10 Abs. 2 GwG is fulfilled.
 - 4.3. As a bank user, I want to fill in the frequency or duration of business relations with the client, so that § 10 Abs. 2 GwG is fulfilled.
 - 4.4. As a bank user, I want to fill in if a bank client or one of their relatives is a politically exposed person so that § 10 Abs. 1 GwG (4) is fulfilled.
5. Revisionsafe documentation of KYC profiles.

User Stories:

- 5.1. As a bank user, I want to save filled-in information as a KYC profile, to provide revision-safe documentation.
- 5.2. As a bank user, I want to recall a KYC profile, to provide revision-safe documentation.

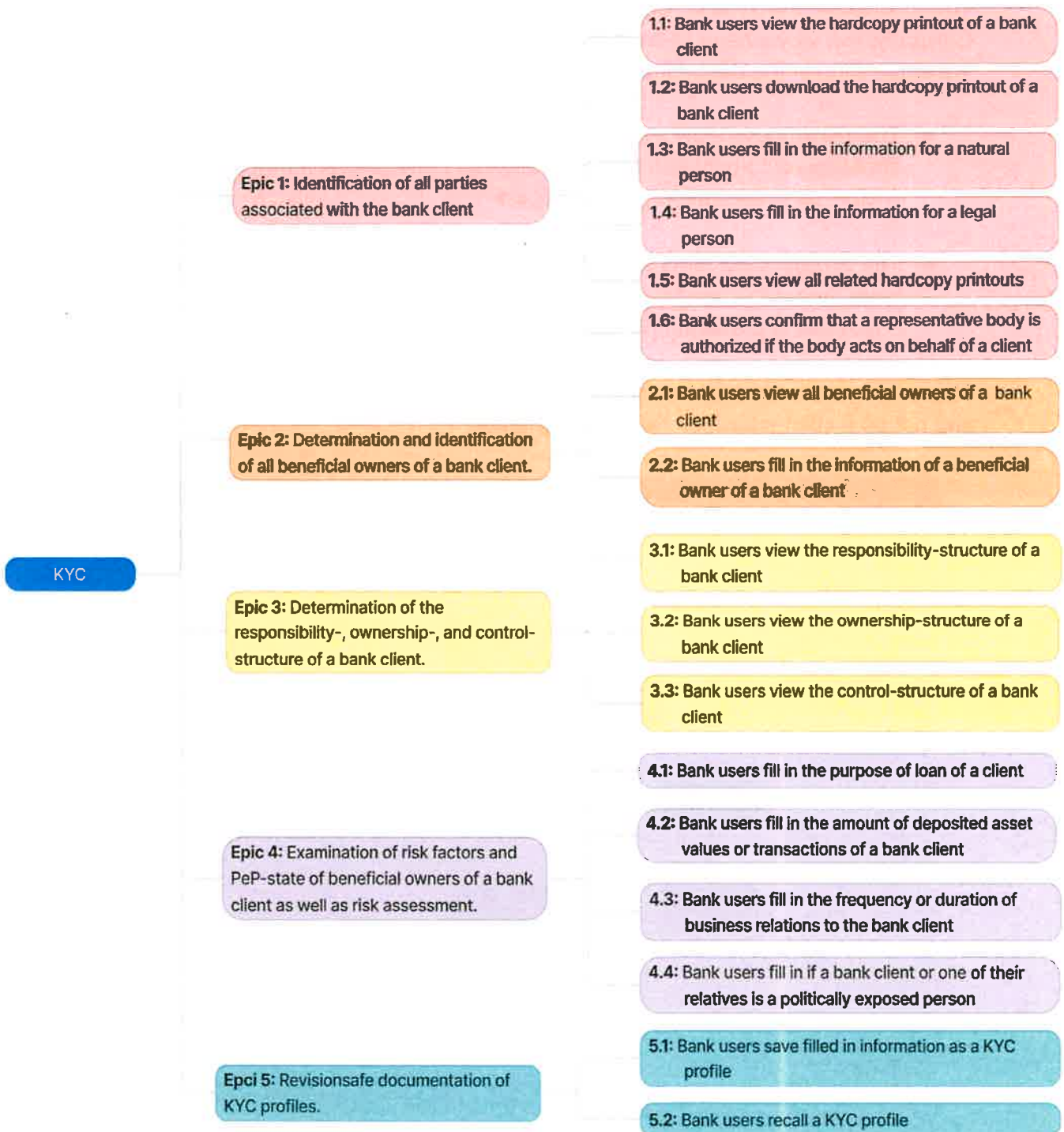


Figure 3.3.1 Epics and user stories of KYC

3.4 User Interface

The application that is to be developed exposes a user interface for interaction. Currently, two views have been identified that aim to enable a user to interact with the application in the desired way:

- The *Dashboard-View* aims to present the relationships (like ownership structure, etc.) between entities in the KYC process as a graph. A Mockup is presented in Figure 3.4.1.
- The *Wizard-View* aims to guide a bank user through the KYC process and have them fill in needed information in forms if necessary. A Mockup is presented in Figure 3.4.2.

The Mockups should only be used as a visual aid when envisioning the application and not as a strict reference for implementation.

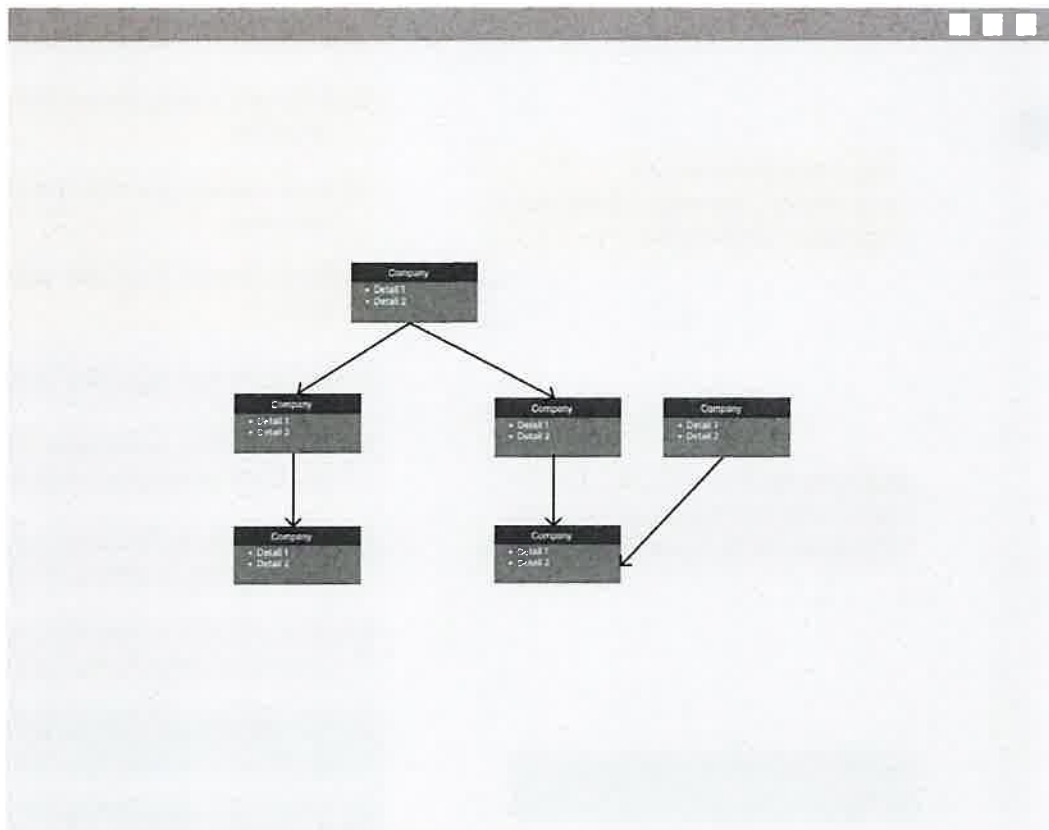


Figure 3.4.1 Mockup - Dashboard-View

The mockup shows a web interface for KYC (Know Your Customer) with a wizard view. On the left is a sidebar with the user's name 'Max Mustermann' and three navigation links: 'Bestand', 'Hilfe', and 'Logout'. The main content area contains two blue-tinted form panels. The top panel, titled 'Unternehmen', has two columns of input fields: 'Name des Unternehmens', 'Ort', 'Steuerland', and 'Gründungsdatum' on the left; and 'Straße und Hausnr.', 'PLZ', 'Rechtsform', and 'Webseite' on the right. The bottom panel, titled 'Personen:', contains two input fields, both labeled 'Gesetzliche Vertreter:'.

Figure 3.4.2 Mockup - Wizard-View

3.5 API Exposure

The frontend and backend communicate via a REST-API. This API should be general enough so it can be used by other microservices in the Neoloan ecosystem. Furthermore, the API should be fully documented with the [OpenAPI](https://www.openapis.org/)² (ver. 3.1.0) specification standard, from which a static HTML page should be generated and then integrated with the rest of the application documentation, to be made available to the client.

² URL: <https://www.openapis.org/>

3.6 Non-Functional Requirements / Environment requirements

The product resulting from this project is going to be implemented as a distributed web application. Details about the environment in which this application is expected to run can be found in [Section 4.1](#). Since the application is a webapp, it exposes a frontend in a browser. The frontend should work as specified on a chromium-based³ browser (ver. >= 108.0.5359.100). To assert this, all tests that involve the user interface should be run on this platform.

There are no further non-functional requirements set at the point of writing this document.

3.7 Acceptance Criteria

The project is subject to many legal regulations, the exact details of which are unknown to the client and the team at the time this document is written.

The derivation of sensible user stories and consequently acceptance criteria from the legal regulations is a key aspect of the project and can only be achieved during the active development of the project. Since it is very difficult to estimate how many of the user stories can reasonably be implemented by the team in the given timeframe, the client and the team agreed that it would be unproductive to set many hard acceptance criteria from the start. Instead, a few hard acceptance criteria are set and the rest is derived from user stories in the way described in [Section 3.1](#) during the active development of the project. The acceptance criteria that are set are listed below.

1. The application is deployed as a Kubernetes cluster on the loft instance provided by the client (details in [Section 4](#)).

Criterion fails, if the application as a whole or any of its components fail to be deployed in a Kubernetes cluster on the provided loft instance, the frontend and backend cannot communicate or the backend cannot establish a database connection.

2. The application exposes a frontend accessible via a chromium-based browser (ver. >= 108.0.5359.100) over the network.

Criterion fails, if no frontend is accessible when the application is running or the frontend application cannot deliver a First Contentful Paint (FCP) because it fails to load.

³ Chromium: <https://www.chromium.org/chromium-projects/>

3. All implemented functionality is accessible via a REST-API.

Criterion fails, if no REST-API is accessible over the network while the application is running or there exists any functionality based on an implemented user story that is not accessible via the API.

4. The REST-API is fully documented adhering to OpenAPI specification in JSON format.

Criterion fails, if there is no JSON document containing the API documentation adhering to the OpenAPI specification in version 3.1.0 on GitLab Pages in the provided GitLab instance (Details in [Section 5](#)).

5. A rendered HTML version of the REST-API documentation is accessible.

Criterion fails, if there is no generated HTML version based on the REST-API documentation in JSON format on GitLab Pages in the provided GitLab instance.

6. The application should be able to display a graph showing the information of the current KYC-Profile

Criterion fails, if it is not possible to display a *correct graph representation* of the information in the current KYC-profile on the frontend. The definition of what a *correct graph representation* is will be decided on during the active development of the project.

7. The application should enable a bank user to start the KYC-Process-Wizard.

Criterion fails, if it is not possible to start a wizard-like application view on the frontend with the press of a button. This view should navigate the user through the KYC process as defined in the user stories.

8. All implemented user stories should adhere to the specified definition of done. ([Section 3.1](#))

Criterion fails, if a user story that is considered as implemented by the team and the client does not adhere to the criteria detailed in the definition of done.

4. Environments

This Section describes the deployment environment of the application in [Section 4.1](#) as well as the environment instances used for development, testing, and production in [Section 4.2](#). Lastly, the tools used during development are listed in [Section 4.3](#).

4.1 Deployment Environment

4.1.1 Containerization

The goal of this project is to develop a webapp in the form of a distributed application. Because the application is distributed, it is not expected to run on a single computer but rather on a computer cluster. To ease the development and deployment process, operating-system-level virtualization is used to host individual parts of the application. For this, the containerization-system [Docker](#)⁴ is used.

This allows each part of the distributed application (backend service, database instance, etc.) to run in its own isolated environment called a container, which is similar to a virtual operating system but not equivalent.

Containerization allows for easy development and deployment of applications since it decouples the software from the hardware. The environment visible to the application remains the same, independent from the actual hardware used to host the containerization runtime.

4.1.2 Container Orchestration

Containerization is a convenient method to create a controlled environment for the individual parts of a distributed application, but in itself does not solve many important aspects associated with the development of distributed applications like networking, load-balancing, and horizontal scaling.

To address these issues and manage the totality of all containers that make up the application, the container orchestration system [Kubernetes](#)⁵ is used. With Kubernetes, it is possible to describe the desired container configuration as well as behaviors for scaling, self-healing, etc., and let it run on a provided set of computers called nodes.

⁴ URL: <https://www.docker.com/>

⁵ URL: <https://kubernetes.io/>

4.1.3 Cluster Virtualization

Kubernetes solves many of the common issues associated with deploying and managing a distributed application, but during the development of such an application, it is desirable to quickly try out different infrastructure configurations and also have multiple cluster setups running at the same time for different purposes (e.g. different developers, production and staging environment). Providing dedicated hardware to all potential cluster setups would be very expensive and resource-inefficient since the machines would sit idle most of the time.

To alleviate these issues the cluster virtualization system [loft](https://loft.sh/)⁶ is used.

With loft, it is possible to run multiple virtual clusters on a set of hardware nodes and quickly instantiate or remove Kubernetes deployments as needed.

4.1.3 Versions

Table 4.1.3.1 lists the versions of the systems described in the previous sections used in this project.

Tool	Version
Docker for Gitlab CI	v18.09.9
Kubernetes	v1.22.9
Loft	v2.3.2

Table 4.1.3.1 Versions of deployment systems

⁶ URL: <https://loft.sh/>

4.2 Environment instances

The client provides access to an installation of loft to enable the instantiation of multiple Kubernetes deployments for development, production and testing purposes. The loft installation runs in the cloud by use of Amazon Web Services (AWS). The specifications of the AWS Elastic Compute Cloud (EC2) instances used for the loft installation are not disclosed to the team but are also not of importance for the intended purposes of this project.

There are no explicit limits set by the client on the resource consumption in the loft installation (CPU hours, Memory, etc.) resulting from this project but it shouldn't be excessive and within reason with regards to the bounds of this project. If at any point the client is of the opinion that resource consumption should be reduced, they inform the team as soon as possible (independent from the sprint cycle) and the team adjusts accordingly.

4.2.1 Development Environment

During development, the team can instantiate temporary Kubernetes deployments using the loft installation access provided by the client. These temporary deployments can be used by the team to manually test out the code they are working on or to try out different infrastructure configurations. Versions of the deployment systems used can be found in [Section 4.1.3](#).

There is no explicit limit set by the client on the amount of instantiated clusters but it should be within reason with regard to team size and the bounds of this project.

There is no explicit limit set by the client on the resources consumed by a Kubernetes deployment but it should be within reason with regard to the bounds of this project.

The client has access to these temporary deployments but they are not to be considered for evaluation of the current state of the project.

4.2.2 Production Environment

The loft installation access provided by the client is used by the team to create a Kubernetes deployment that shall be considered the production environment.

Versions of the deployment systems used can be found in [Section 4.1.3](#).

This deployment is continuously updated using a continuous integration (CI) pipeline hosted on the client-provided GitLab instance to show the current status of the project.

There is no explicit limit set by the client on the resources consumed by this Kubernetes deployment but it should be within reason with regard to the bounds of this project.

The client has access to the production environment and can use it to evaluate the current state of the project.

4.2.3 Testing Environment

When integration tests are run by the CI pipeline, a temporary Kubernetes deployment is used similar to a development environment. Details of this can be found in [Section 4.2.1](#).

4.3 Tools

The tools used by the team during the development are listed in Table 4.3.1.

Problem	Tool Version
Version control / SCM	GitLab 15.6.1
GUI Design	Figma 9.0
Time tracking	Excel template -
Issue tracking / Projektmanagement	Jira <i>atlassian cloud</i>
IDE	Visual Studio Code 1.74
Internal communication	Discord -
Communication with client	Slack -
Team collaboration	Confluence <i>atlassian cloud</i>
Frontend framework	Angular 15
Backend framework	NestJS 9.2.1
Monorepo tooling / Build system	Nx Workspace 15.3.3
Database management System	Neo4j 5.3
Programming Language	Typescript 4.9.4

Table 4.3.1 Tools used during development

5. Deliverables

At the end of the project the team is expected to have produced certain artifacts that contain the implementation of the described project. In the following subsections, the form of these artifacts is described, as well as their method of delivery.

5.1 Source Code

Because the project is not expected to have produced a full production-ready application by the end of it, the source code is the most important deliverable since it enables the client to further develop the project.

All source code (this includes frontend and backend), along with all configuration files that describe deployment details as well as the Continuous Integration (CI) Pipeline is expected to be versioned in a single repository (Monorepo) hosted on an instance of the [GitLab](#)⁷ Source Code Management (SCM) system provided by the client. The client has full read-and-write access to this repository during and after the project which makes an explicit delivery step at the end of the project obsolete. Requirements of the code that is to be delivered can be found in [Section 3.1](#).

5.2 Documentation

The source code for the frontend and backend is expected to be documented by the use of in-code documentation in a reasonable amount and in accordance with the requirements set in [Section 3.1](#). Broad architectural decisions will be described in the company's internal [confluence](#)⁸ which already provides a possibility for version control. The documentation generation tool [compodoc](#)⁹ shall be used to generate a static website (HTML, js, CSS) from the source code as part of the CI pipeline of the project. This static website is expected to be made accessible using [GitLab Pages](#)¹⁰, the deployment of which shall also be part of the CI pipeline. Furthermore, the REST-API documentation mentioned in [Section 3.5](#) should also be made accessible using GitLab Pages.

The client has full access to the generated documentation during and after the project so an explicit delivery step at the end of the project is obsolete. At the end of the project, the documentation is expected to give a comprehensive overview of the implementation details of the project that is sufficient for further development.

⁷ URL: <https://docs.gitlab.com/>

⁸ URL: <https://www.atlassian.com/de/software/confluence>

⁹ URL: <https://compodoc.app/>

¹⁰ URL: <https://docs.gitlab.com/ee/user/project/pages/>

5.3 Kubernetes / Docker

The project is expected to implement a web app that is deployable in a computer cluster as a distributed application. To achieve this, the containerization tool [Docker](https://www.docker.com/)¹¹ and the orchestration tool [Kubernetes](https://kubernetes.io/)¹² are to be used. This deployment is part of the CI pipeline of the project, further details of which are described in [Section 4 Deployment Environment](#).

As mentioned in [Section 5.1](#) the deployment configuration files are to be versioned alongside the source code, which the client has access to. As soon as the project is capable of producing a running application, a working version is expected to be made available to the client through deployment by the CI pipeline on infrastructure provided by the client.

This installation is expected to be continuously updated to represent the current state of the project. Because the client has full access to the running application during and after the project, an explicit delivery step is obsolete. It has not been decided yet whether the application needs to adhere to a specific versioning scheme like semantic versioning, this will be the subject of future discussion between the client and the team.

¹¹ URL: <https://www.docker.com/>

¹² URL: <https://kubernetes.io/>

6. Risk Mitigation

In the following sections potential risks of the project are discussed, as well as strategies to avoid them.

6.1 Risk Matrix

A risk matrix is used to assess the severity of a risk. First risk likelihood and risk impact are assessed, to then compute the risk severity. The risk matrix¹³ (shown in Figure 6.1.1) shows the likelihood of the risk event occurrence on the X-axis and the potential impact that the risk event will have on the project on the Y-axis.

If a task has a risk that is “possible” and “catastrophic” or “almost certain” and “critical” or “almost certain” and “catastrophic”, the team will try to prioritize its avoidance highly and for less likely or less severe, the team will prioritize its avoidance less. This strategy is applied to keep the team from putting too much work into mitigating risks that are very unlikely or not severe and lets it focus on the risks that need more attention.

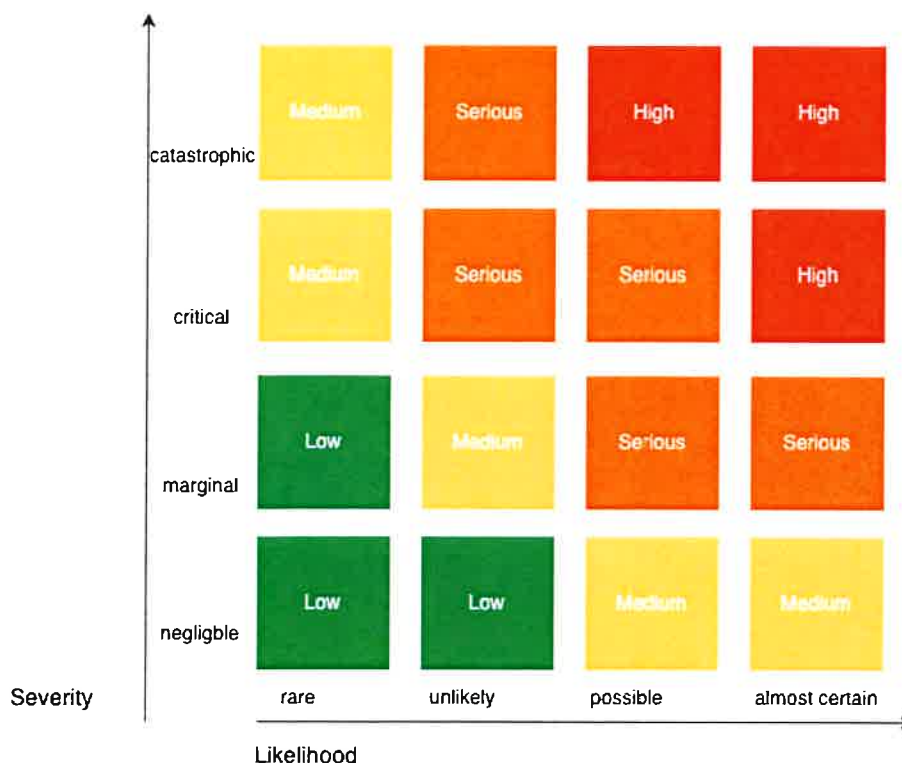


Figure 6.1.1 Risk Matrix

¹³ Bundesamt für Sicherheit in der Informationstechnik, from <https://www.bsi.bund.de/dok/10990692>

6.2 Risks and Risk Avoidance

6.2.1 Absence of a Responsible Person

Situation: A Team member leaves the project or is unexpectedly unavailable and cannot be replaced by another team member.

Severity: critical

Occurrence probability: almost certain

Risk: **high**

Reason: Although the team currently rates its working climate as very harmonious, experience has shown that this risk is usually the main reason for the failure of comparable university projects, which is why the team will prioritize minimizing it. Consequently, the team rates the risk as very high severity and uses high priority to avoid it.

Minimization Strategy: The team employs a code review process to ensure that there are always multiple people familiar with the code and that the success or failure of our project does not depend on a single person. In addition, the team specifically focuses on the development of a positive working atmosphere.

6.2.2 Illness or Vacation

Situation: It may be that one or more people (including the client's representative) become ill or go on vacation and are therefore temporarily unable to continue their work.

Severity: negligible

Occurrence probability: possible

Risk: **medium**

Reason: The likelihood of something like this happening is relatively high (especially considering the current pandemic situation), but the resulting damage can be avoided. Consequently, the team rates the risk as medium risk and uses medium priority to avoid this risk.

Minimization Strategy: The team tries to have multiple team members working on each part of the application to effectively distribute the resulting workload in the event of a failure. Furthermore, the team uses a shared Excel document in which team members can plan and register vacations.

6.2.3 Communication Problem

Situation: Communication problems with the client that result in their wishes not being fulfilled.

Severity: negligible

Occurrence probability: rare

Risk: low

Reason: The representative of the client is a software developer himself and has the appropriate experience with communication in such a project. The same applies to several of the team's members, which is why the team considers it relatively unlikely that such problems will arise. Consequently, the team rates the risk as low risk and uses low priority to optimize it.

Minimization strategy: The team and the client join in regular meetings in which past work is reviewed and requirements accordingly adapted.

6.2.4 External Delays

Situation: Access to external API gets delivered too late by the client.

Severity: negligible

Occurrence probability: rare

Risk: low

Reason: This is unlikely to happen, and it will not cause too serious consequences. A reason for this could be that the client's development process takes unexpectedly long and the client can't provide the API in time. The team and client rate the risk as low risk and use low priority to minimize this risk.

Minimization strategy: The team will contact the client early so they will have a time buffer to provide the API to the team.

6.2.5 Failed External API

Situation: External API does not work correctly: The API is unstable.

Severity: medium

Occurrence probability: unlikely

Risk: medium

Reason: An error occurs when using external APIs provided by the client. This is unlikely to happen. The team rates the risk as medium risk and priority also as medium.

Minimization-strategy: The team will choose stable versions of APIs and will not choose APIs with unstable factors.

6.2.6 Unforeseen Bugs

Situation: Hard-to-solve bugs occur and can not be fixed right away

Severity: marginal

Occurrence probability: possible

Risk: **serious**

Reason: Experience shows that many bugs appear during the development. This can lead to a lot of development time being used for debugging and less development time being used to develop new features.

Minimization-strategy: Mitigation of this is possible by testing and especially writing automated regression/unit tests.

6.2.7 Knowledge Gap in the Team

Situation: There is a knowledge gap in the team so some requirements can not be implemented.

Severity: negligible

Occurrence probability: unusual

Risk: **low**

Reasons: Due to previous projects our team has already experience in software development projects. The team already has some experience with the technology that we are using. This leads to an evaluation of "low" for this risk.

Minimization-strategy: The team tries to find out how to implement the project by spending time sharing development experience and doing workshops and getting in contact with the client if necessary.

7 Legal Regulations

Both the client and the contractor are located in Germany during the time of the project, therefore the project is subject to German law. To ensure that the following sections about legal regulations are valid under German law, they are written in German instead of English.

7.1 Vertragspartner

7.1.1 Auftraggeber

Neoloan AG

Meglingerstr. 20
81477 München

7.1.2 Auftragnehmer

Studentische Arbeitsgruppe NeoTheOne

Technische Universität Darmstadt
Fachbereich Informatik
Software Technology Group
Hochschulstraße 10
64289 Darmstadt

Die Studentische Arbeitsgruppe NeoTheOne besteht aus folgenden Personen:

- Lea Pellikan
- Felix Wortmann
- Leon Chemnitz
- Chen Yang
- Lei Zhuang

7.2 Vertragsgegenstand

Das Vertragsverhältnis zwischen Auftraggeber und Auftragnehmer entspricht einem Werkvertrag. Gegenstand des Vertrages sind die in [Abschnitt 5](#) beschriebenen, durch den Auftragnehmer zu erstellenden Artefakte, nachfolgend zusammenfassend als Software bezeichnet. Die Erstellung der Software erfolgt nach den in Abschnitten [3](#) und [4](#) beschriebenen Spezifikationen in einem Modus Operandi beschrieben in [Abschnitt 2](#). Dieses Dokument bildet durch Unterzeichnung durch Auftragnehmer und Auftraggeber die Basis dieses Vertrages.

7.3 Nutzungs- und Urheberrechte

Urheber der Software ist nach §7 UrhG der Auftragnehmer, also die studentische Arbeitsgruppe. Der Auftragnehmer überträgt nach §31 UrhG sowie insbesondere § 69 c UrhG dem Kunden sämtliche im Rahmen dieses Vertrages und seiner Erfüllung entstandenen Nutzungsrechte. Der Kunde ist berechtigt, die ihm übertragenen Rechte an Dritte zu deren freier und uneingeschränkter Verwendung weiterzugeben. Die Urheberpersönlichkeitsrechte des Auftragnehmers bleiben unberührt.

7.4 Schadensersatz- und Wartungsansprüche

7.4.2 Gewährleistung und Wartung

Gewährleistung jeglicher Art für die entwickelte Software wird sowohl durch die studentische Arbeitsgruppe, als auch durch die Technische Universität Darmstadt ausdrücklich ausgeschlossen. Es ist dem Auftraggeber bekannt, dass trotz Erprobung unter repräsentativen Einsatzbedingungen, bei Softwareprodukten Fehler nicht gänzlich auszuschließen sind. Weder die studentische Arbeitsgruppe noch die Technische Universität Darmstadt übernehmen Wartungsverpflichtungen jeglicher Art für die erstellte Software.

7.4.2 Schadensersatzansprüche

Das Risiko für Schäden jeglicher Art, die durch die Nutzung der Software entstehen können, liegt beim Auftraggeber. Weder die studentische Arbeitsgruppe noch die Technische Universität Darmstadt haften für durch die Nutzung der Software entstandene Schäden. Der Auftraggeber trägt allein das Risiko für Schäden wie z.B. Datenverluste, Hardwaredefekte oder Systemabstürze. Etwaige durch die Benutzung der Programme entstehende Kosten für die Wiederbeschaffung von Daten und Programmen sowie Kosten für Reparaturen an der Hardware sind vom Benutzer zu tragen. Der Auftragnehmer kann nicht für Sach-, Personen- und Vermögensschäden haftbar gemacht werden, die im Zusammenhang mit dem Einsatz der vom Auftragnehmer erstellten Software entstehen könnten. Bei Schaden Dritter, der durch den Einsatz des Produktes entsteht, übernimmt der Auftragnehmer keinerlei Haftung. Eine Haftung der Technischen Universität Darmstadt für den Auftragnehmer als deren Erfüllungsgehilfen nach §278 BGB scheidet ebenfalls aus.

7.5 Vertraulichkeitsvereinbarung

Alle in [Abschnitt 5](#) beschriebenen Artefakte, sowie alle weitere, im Rahmen des beschriebenen Projektes entstandenen vertraulichen Informationen sind nachfolgend zusammenfassend als Geschäftsgeheimnis bezeichnet.

Der Auftragnehmer ist nicht berechtigt, das Geschäftsgeheimnis entgegen der Vereinbarung zu nutzen oder offenzulegen. Offenlegung bezeichnet das Eröffnen des Geschäftsgeheimnisses gegenüber einem Dritten. Offenlegung bedeutet nicht Öffentlichkeit. Vertrauliche Informationen im Sinne dieser Vereinbarung sind sämtliche Informationen (ob schriftlich, elektronisch, mündlich, digital verkörpert oder in anderer Form), die von dem Auftraggeber an den Auftragnehmer im Rahmen des Projektes offenbart werden.

Der Auftragnehmer verpflichtet sich, das Geschäftsgeheimnis streng vertraulich zu behandeln und nur im Zusammenhang mit dem Projekt zu verwenden. Diese Verpflichtung bleibt bestehen, auch nachdem das Projekt und das damit verbundene Arbeitsverhältnis beendet ist.

Eine Offenlegung des Geschäftsgeheimnisses vom Auftragnehmer gegenüber Vertretern der Technischen Universität Darmstadt ist zweckgebunden gestattet.



7.6 Anwendbares Recht und Gerichtsstand

Für die Rechtsbeziehungen zwischen Auftraggeber, Auftragnehmer, Technische Universität Darmstadt und etwaigen Dritten gilt das Recht der Bundesrepublik Deutschland als zwingend vereinbart. Andere nationale Rechte außerhalb der Bundesrepublik Deutschland werden ausgeschlossen. Der Gerichtsstand ist Darmstadt.

7.7 Abnahme

Das agile Vorgehen des Projektes ermöglicht es, die Qualität der Softwareteillieferungen (Sprints) zeitnah sicherzustellen und abzunehmen. Dies wird durch Zwischenabnahmen von Sprints sichergestellt.

Eine finale Abnahme folgt zum Abschluss des Projektes am 20.03.2023. Nach diesem Datum gelten das Projekt und das damit verbundene Arbeitsverhältnis als beendet.

Ort / Datum	<u>München, 21.12.22</u>	Ort / Datum	<u>Darmstadt / 16.12.2022</u>
Neoloan AG	<u></u>	NeoTheOne	<u>Lea Pellikan</u>
			<u>Prof. Dr. Sam</u>
			<u></u>
			<u>Chunfeng</u>
			<u>Lei Zhuang</u>

