Hochschule für Technik
und Wirtschaft Berlin

University of Applied Sciences

Design & Implementation of a Fraud Detection System for Autonomous Teams (Total pages should be: 50 [without Attachments])

Abschlussarbeit

zur Erlangung des akademischen Grades:

Bachelor of Science (B.Sc.)

an der

Hochschule für Technik und Wirtschaft (HTW) Berlin
Fachbereich 4: Informatik, Kommunikation und Wirtschaft
Studiengang *Angewandte Informatik*

1. Gutachterin: Prof. Dr. Christin Schmidt
2. Gutachter: MSc. Tobias Dumke

Eingereicht von Louis Andrew [s0570624]

Datum

Abstract

[Summary of the thesis]

Contents

List of Figures

List of Tables

Listings

1. Introduction

Vorliegendes Template enthält exemplarisch (und damit unvollständig) Gliederungspunkte, Bestandteile und Hinweise für ein typisches Softwareentwicklungsprojekt, bei dem ein Prototyp erstellt wird. Es dient als Hilfestellung zu Ihrer weiteren Verwendung. Selbstverständlich müssen Sie selbst weitere Ergänzungen und Anpassungen vornehmen.

Viel Erfolg sowie gutes Gelingen bei Ihrer Abschlussarbeit!

Der Textteil beginnt hier und wird arabisch mit dieser Seite beginnend mit »1« arabisch nummeriert. Der Textteil gliedert sich in Kapitel und Unterkapitel. Soll jede Hierarchieebene benannt werden, dann ist folgende Terminologie üblich:

- 1. Hierarchieebene: Chapter
- 2. Hierarchieebene: Section
- 3. Hierarchieebene: Subsection
- 4. Hierarchieebene: Subsubsection

Der inhaltliche Aufbau einer Abschlussarbeit im Studiengang *Angewandte Informatik* hängt selbstverständlich vom Thema und vom Inhalt ab. Abweichungen von der diesem Template zu Grunde liegenden Gliederungsstruktur sind immer möglich, manchmal sogar zwingend notwendig. Stimmen Sie sich diesbezüglich immer mit Ihren Gutachter(inne)n ab.

Vergessen Sie niemals, all Ihre verwendeten Quellen anzugeben und korrekt zu zitieren[1]. Quellen können manuell referenziert und im Quellenverzeichnis eingetragen werden. Ergänzend bieten viele Textverarbeitungsssteme auch ausgelagerte Quellenverwaltungsdateien und - systeme an, über die mittels entsprechender Befehle im Textteil zitiert werden kann[2].

Visualisieren Sie im Textteil angemessen, z.B. mittels Abbildungen und Tabellen. Vorliegendes Template enth"alt beispielhaft eingebundene Abbildungen und eine Tabelle (vgl. f.), welche der Steinlausforschung[3] entnommen sind.

[1] Ergänzende Informationen können Sie auch in eine Fu"snote auslagern. Hier wird die Fu"snote dazu genutzt, um Ihnen bei Interesse am Thema Zitation vertiefende Quellen (z.B. [1] oder [3]) anzubieten.

[2] Wie Sie hoffentlich feststellen werden, erfolgt die Literaturverwaltung in diesem Template mittels einer *.bib-Datei (diese enthält die verwendeten Quellen), welche die *.tex-Datei mittels Verwendung von biblatex und bibtex ergänzt.

[3] Analog zu Straube (In: [13]) handelt es sich bei der Steinlaus (*petrophaga lorioti*) um das »*kleinste einheimische Nagetier*«. Als stimmungsaufhellender Endoparasit erreicht es eine Grö"se von ca. 0,3 bis 3 mm und stammt aus der Familie der Lapivora. Die Steinlaus kommt ubiquitär vor und ist in der Regel apathogen.

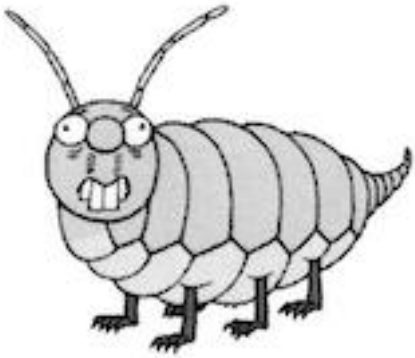

Figure 1.1.: *Example image: Who is Steinlaus?; Bildquelle [10]*

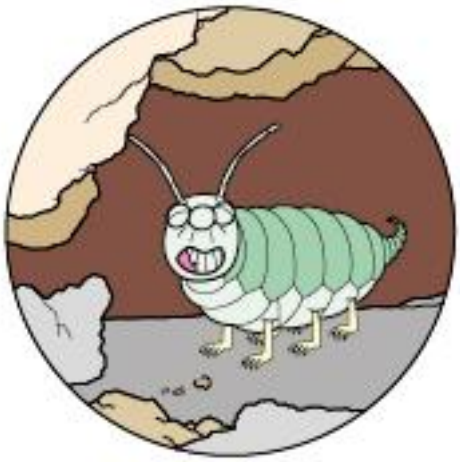

Figure 1.2.: *Beispielgrafik: Fressende Steinlaus; Bildquelle [9]*

Table 1.1.: *Übersicht: Untersuchte Steinläuse*

Untersuchte Objekte mit Lokation des Habitats		
ID (nickname)	Ort	Grö"se/Länge (in mm)
1 (Rosalinde)	Berlin, Mauerpark	1.4
2 (Devil in disguise)	Brandenburg, BER-Airport	2.8
3 (Hannes)	Berlin, Olympia-Stadion	2.1
4 (Her Majesty)	Berlin, Humboldt-Forum	2.0

1.1. Background and Motivation

The background and motivation of the thesis is to get my bachelor degree.

1.2. Goal

Goal of the thesis is to build a cool software

1.3. Scope

Scope of the thesis is to not build a new internet protocol

2. Fundamentals

TODO! [Beschreibung des Kontextes der Arbeit mit allen durch die Problemstellung tangierten Bereichen, Methoden, Theorien, Erkenntnissen, Technologien, ...]

2.1. Kontext

2.1.1. Domain

2.1.2. Technologien

2.1.3. Methoden und Konzepte

2.2. ...

2.2.1. ...

2.2.2. ...

3. Requirement Analysis (!)

[Beschreibung der Erhebung, Granularisierung und Priorisierung der zu Grunde liegenden Anforderungen]

3.1. Goal?

Hi mom, I'm . trying to cite ISO thingy [7]

3.2. Application Environment

3.3. Analysis / State of Art

3.4. Requirements

Optionals:

- *Rahmenbedingung*
- Methods

4. Conception and Design

[Beschreibung des Entwurfs auf Basis der Methodologie / der geplanten Vorgehensweise zur Problemlösung im Kontext der Anforderungen (i.A. der Art der Arbeit)]

4.1. System Architecture

A system architecture diagram is created to help to understand the system as well as its components better.

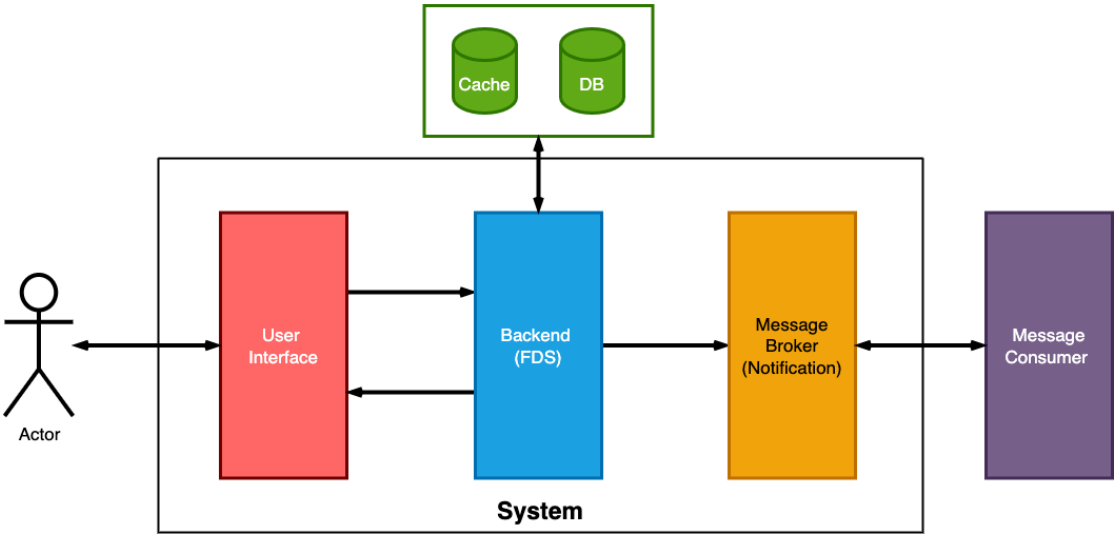

Figure 4.1.: *System architecture diagram*

The system diagram visualizes the components of the system and their interaction with each other. Internally, the system contains 3 independent components; user interface (UI), fraud detection service (FDS) and a message broker (notification system). Externally, the system would interact with a database and optionally a cache memory[1] to persist information needed for the validation process. Apart from the internal components of the system and its connection to the database, the diagram also visualizes an external system (*message consumer*) which indeed is out of the system's scope, but plays an important role to determine which action to be taken on certain events. Further information on the function as well as connection between each component will be discussed on the next section.

[1]Cache memories are small, but extremely fast memory used in computer systems to store information that are going to be accessed in a small timeframe [14]

4.2. Software Architecture

As the requirement is clear and the components of the system are defined, a software architecture is needed. A software architecture plays an important role in a software development project by providing a structure on how the software should be built and decisions made during this stage would be vital for the development process going forward. As Garlan wrote in one of his work *Software Architecture*, a software architecture "plays a key role as a bridge between requirements and implementation."[6] A software architecture diagram is therefore created to help visualize the structure, functions and role of each component of the system.

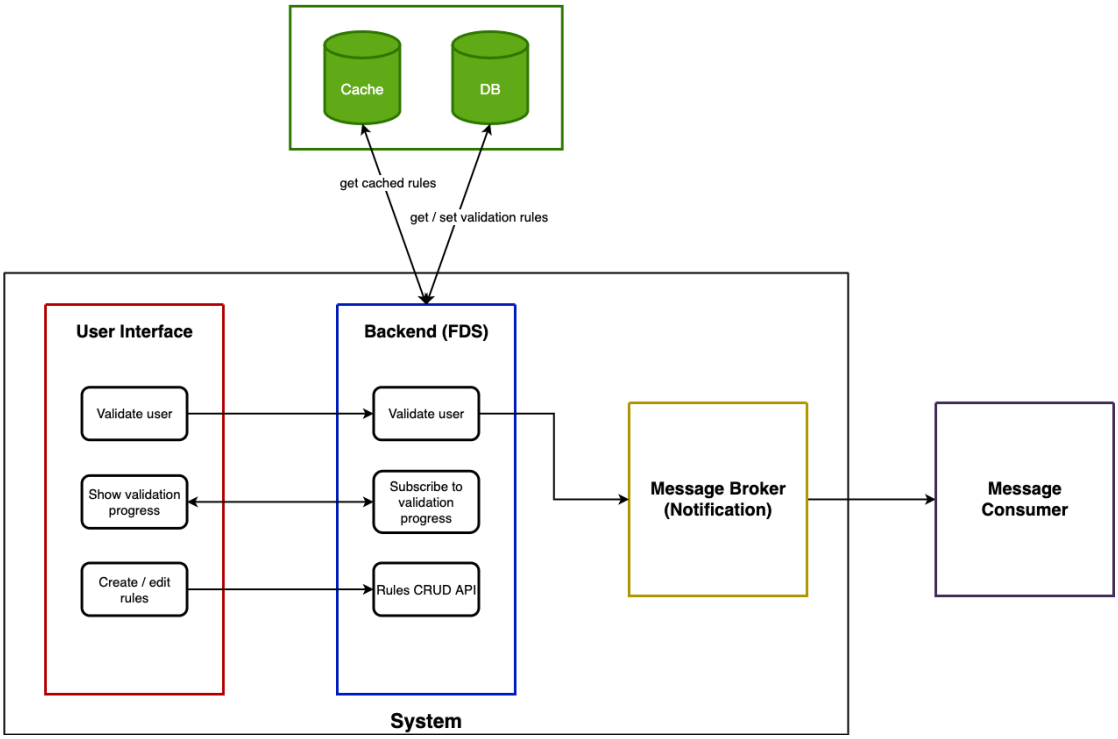

Figure 4.2.: *Software architecture diagram*

In a real world production environment, the user interface might need to be separated into several independent applications. A dedicated app to manage validation rules should be reachable only for internal employees (such as a developer from the company) while a customer facing UI that contains a registration form could also trigger a validation process directly after a new registration. An additional UI to display the progress of the currently running validation processes could also be built specifically for customer service agents, so that a fast reaction on certain suspicious customers can be done. For this project, all the use cases mentioned above will be implemented and combined into a single web application.

The fraud detection system (*backend, FDS*) is the core engine of the system where validation processes would be run. The FDS is responsible for all CRUD operations regarding the validation rules. User should be able to create, read, update and delete validation rules via an HTTP request. A detailed explanation on validation rules will

be discussed in subsection 4.5.1. A database connection should be established on the FDS to persist the validation rules. An optional connection to a cache memory could also be established to enable a faster access to the data needed.

The core functionality of the FDS is to run validation process of a customer by evaluating a collection of validation rules in relation to a given customer data. The execution of the validation process could be scheduled via a single HTTP request that contains the customer data on its request payload[2]. A validation process is run asynchronously, the FDS will not return the result of the validation directly as a response to the HTTP request. This is intended to prevent a slow response time of the FDS.

Clients could then subscribe to the latest progress of a validation process by accessing an additional endpoint provided by the FDS. A subscription mechanism will be implemented to prevent the need of a request polling on the client side, either by using the WebSocket protocol[3] or something similar.

After a validation process is completed, the FDS should return the result of a validation and further actions should be handled by external services out of the system's scope. This separation of concern is intended to decouple the execution of a validation process and the processing of its result. As there might be several implications on what a validation result might mean, the validation result will be distributed across multiple clients. To achieve this functionality, the Observer[5, pp. 293-303] design pattern will be implemented, and a messaging system is needed to act as a bridge between the message producer and its consumers. In this specific architecture, the FDS will act as a message producer, producing a message containing the validation result to the message broker whenever a validation process is done and the external services will act as message consumers, by consuming a message queue created by the message broker and running actions on certain cases independently.

4.3. Technologies

The software architecture determines not only the structure of the software, but it also helps in defining which type of technologies might be needed to build the system as efficiently as possible. Different technologies have their own advantages and disadvantages, and the goal of this phase is not to find the best technology or the best programming language, but rather to find the most suitable set of technologies given the priorities of the project and preferences of the writer. From the software architecture diagram listed on Figure 4.2, the technologies to be used on the following components should be defined:

- User interface (web application)
- FDS (server-side application)
- Message broker / notification system

[2] A request payload refers to data sent by a client to the server during an HTTP request, usually attached to the request body[2, section "4.3 Message Body"]

[3] In [12], Fette and Melnikov introduced the WebSocket protocol as a way to establish a two-way communication between a browser-based client and a remote host without relying on opening multiple HTTP connections.

- Database and caching memory

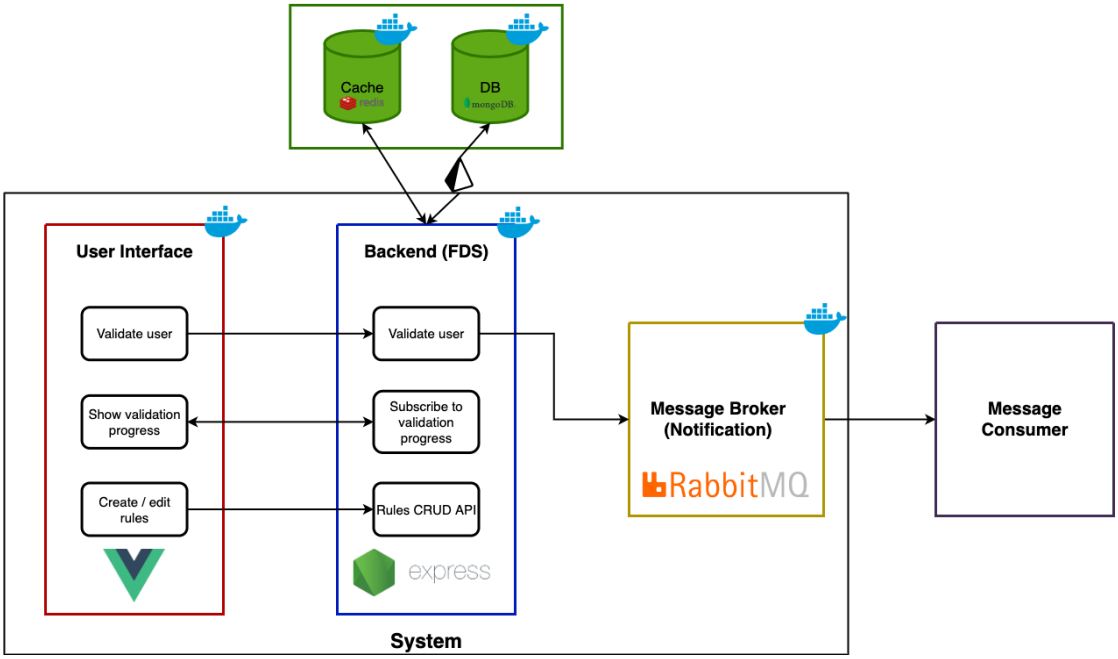

Figure 4.3.: *Software architecture diagram with technologies used*

The previous software architecture diagram is therefore extended with additional logos of the technology used for each component of the system. All internal components of the system should be run as a Docker[4] container. Running the applications as a Docker container means that each application is started and run in isolation, ensuring portability to other operating systems. The database and caching memory will also be run as a Docker container, to avoid the need of installing the dependencies needed and to enable the possibility to start all the components of the system using a single command with Docker Compose[5].

4.3.1. User Interface

The technology used to build the user interface is VueJS (3. Version, also known as *Vue3*)[6], a JavaScript framework built on top of HTML, CSS and JavaScript for building a reactive user interfaces using a component-based programming paradigm. To ensure type safety, the user interface is built with TypeScript[7], rather than plain JavaScript, which is also supported by Vue3.

[4] *Docker* is an open source software used to containerize applications in a package with its dependencies and operating system, making it runnable in any environment. Homepage: https://www.docker.com/

[5] *Docker Compose* is an open source tool for running multiple Docker containers. GitHub repository: `https://github.com/docker/compose`

[6] *Vue3* is an open source JavaScript framework to build user interfaces. GitHub repository: `https://github.com/vuejs/core`

[7] *TypeScript* is an open source programming language built by Microsoft on top of JavaScript by adding additional optional static typing. A TypeScript program will be compiled to a plain JavaScript program, before being executed in environment such as browser or NodeJS environment. GitHub repository: `https://github.com/microsoft/TypeScript`

4.3.2. FDS

The technology chosen to build the FDS is Node.JS[8]. Node.JS is chosen not only because the writer is familiar with it, but also the event loop architecture of Node.JS enables the possibility to perform non-blocking I/O operations asynchronously. Each validation process will be an asynchronous process, which wouldn't block the main thread of the application. To ensure type safety, TypeScript is also used here rather than plain JavaScript. Express.JS[9] is the web framework of choice to build the FDS. Express.JS provides a simple and declarative API to build a web application with ease and speed. An object-relational mapping tool (ORM) is used in this application to provide an easier access to the database, and additionally to keep the database schema in sync between the database server and the FDS application. The ORM of choice for the application is Prisma[10], as it provides a straightforward integration with TypeScript, generating TypeScript types automatically from the database schema.

4.3.3. Message Broker / Notification System

A reliable message broker is needed to make sure that all validation result actually reaches the consumers. The technology chosen for this component is RabbitMQ[11], as it is not only reliable, but also has an easy guide to set up as well as a big collection of client libraries for multiple programming languages.

4.3.4. Database and Caching Memory

A database is needed to store data regarding validation rules. Each database system has their own use cases and weaknesses. For this particular project, MongoDB[12] will be used as the database system of choice. Redis[13] is chosen as the technology of choice for the caching memory because of its simple API and reliability.

4.4. System Sequence

To better visualize the interaction between components of the systems as well as its concrete functionalities, sequence diagrams for specific use cases are created. Each specific use cases are represented in a single sequence diagram, making it easier to convert the use case into corresponding technical user stories.

[8] *Node.JS* is an open source JavaScript runtime environment that runs on Google's V8 engine, enabling JavaScript programs to be run out of the browser environment. GitHub repository: `https://github.com/nodejs/node`

[9] *Express JS* is an open source web application framework for Node.JS. GitHub repository: `https://github.com/expressjs/expressjs.com`

[10] *Prisma* is an open source ORM for Node.JS and TypeScript. GitHub repository: `https://github.com/prisma/prisma`

[11] RabbitMQ is a messaging broker, enabling the distribution of messages across multiple clients. RabbitMQ homepage: `https://www.rabbitmq.com/`

[12] *MongoDB* is a source-available NoSQL database developed by MongoDB Inc. MongoDB homepage: `https://www.mongodb.com/`

[13] *Redis* is an open-source in-memory data structure store. GitHub repository: `https://github.com/redis/redis`

4.4.1. Customer Validation on a Registration Event

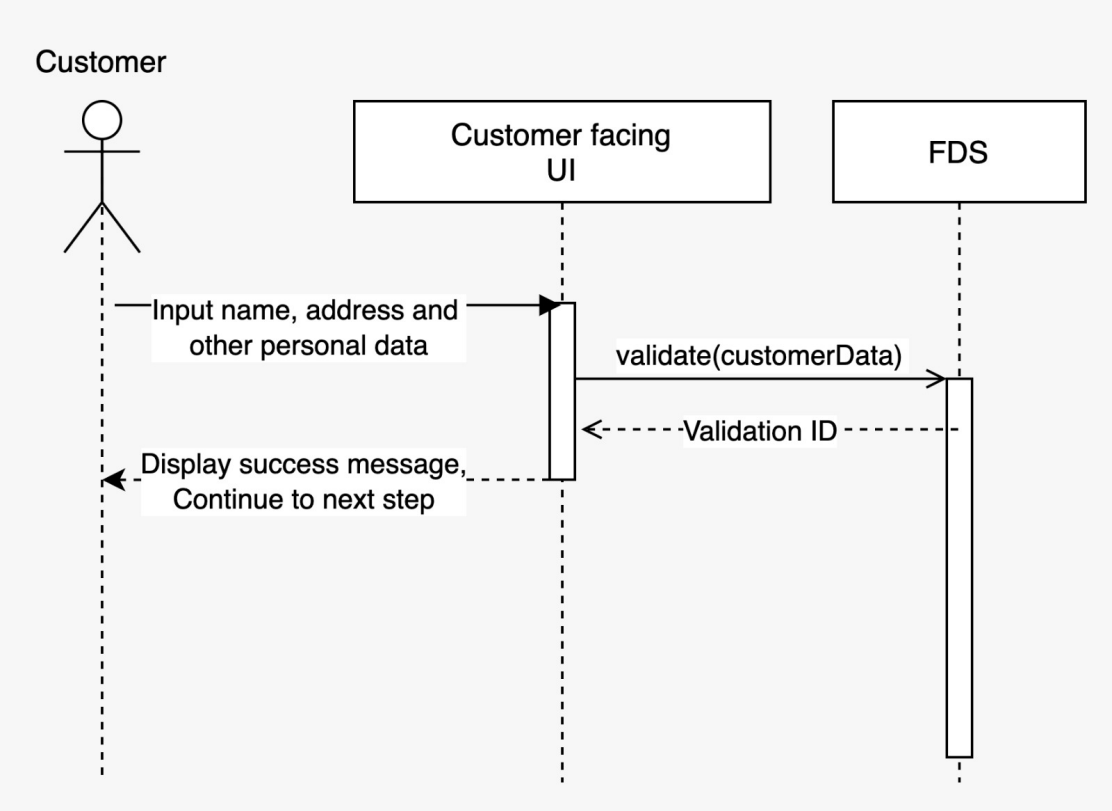

Figure 4.4.: *System sequence diagram for a customer validation when a new customer is registered*

A system sequence can be defined by analyzing the following use case:

> "As a stakeholder, I want to verify users, so that the company can have more confidence that the existing user base is trustworthy"

By further analysis of the use case listed above, the following sequence will be executed by the system:

- A new customer inputs his or her personal data to a customer facing UI and clicks the *"Register"* button
- The customer facing UI makes an HTTP Post request to the FDS, containing the user's personal data on its request payload
- The FDS receives the HTTP request, and schedules a new validation process to be executed asynchronously
- The FDS responds to the HTTP request by returning a validation ID pointing to the scheduled validation process
- Customer facing UI shows a success message and continues registration to the next step while the validation process runs

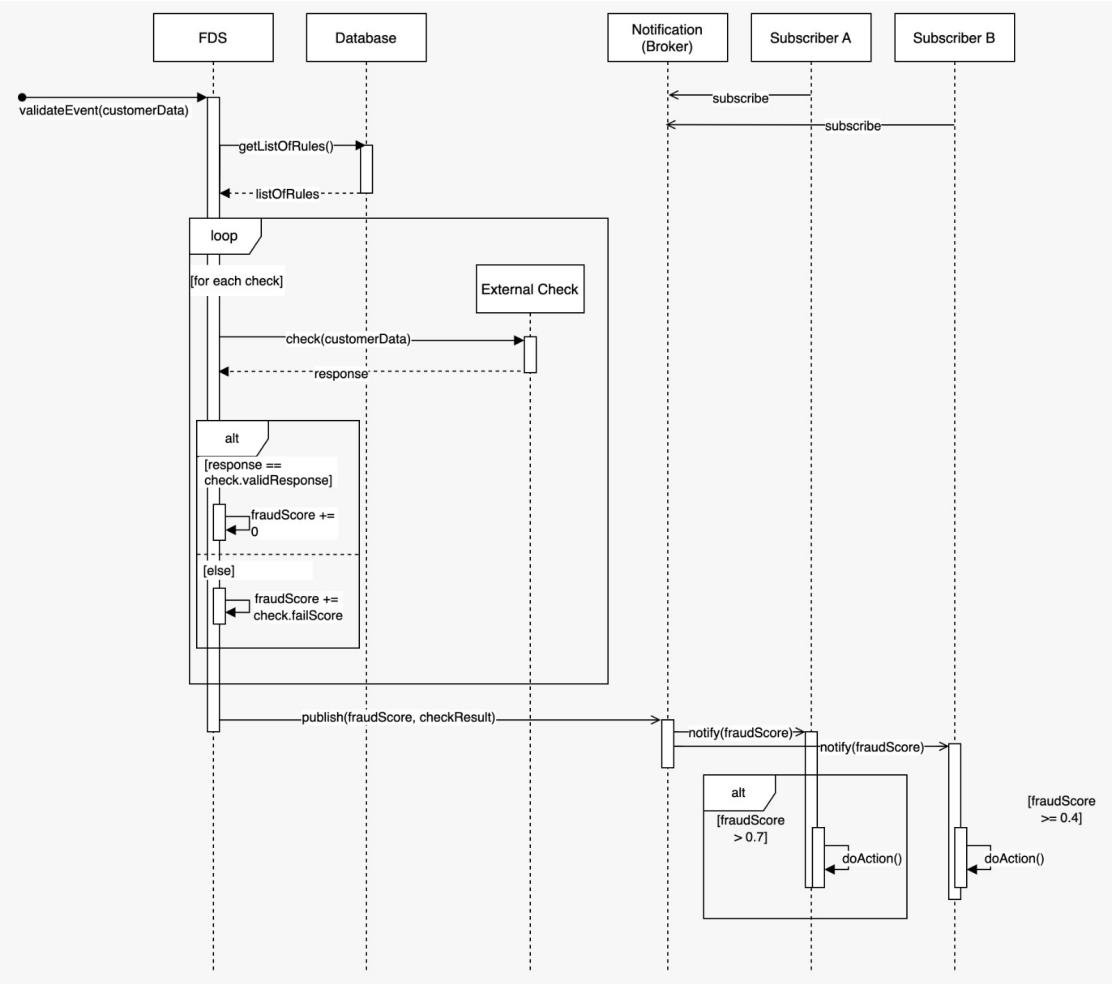

Figure 4.5.: *System sequence diagram for notifications on suspicious cases*

4.4.2. Notification on Suspicious Cases

To fulfill the requirements listed on **TODO: Link Analysis**, a further examination of the following use case should be done:

> "As an employee, I want to be notified when a user seems suspicious, so that I can do necessary actions accordingly"

As a result, the following sequence will be executed by the system:

- The FDS receives an HTTP request to schedule a validation process and responds by returning the ID of the validation process
- FDS retrieves a list of validation rules from the database
- FDS begins to initiate a validation process by setting the fraud score to 0 and looping through the list of validation rules for evaluation
- A validation rule will be evaluated by making an HTTP request to the external endpoint defined by the validation rule and evaluating its response according to the condition specified

- If the response matches all the conditions specified by the validation rule, the rule evaluation will be considered as a success and the fraud score will be incremented with 0. Otherwise, the rule evaluation will be considered as a failure and the fraud score will be incremented by the *fail score* specified by the validation rule
- After the evaluation of all validation rules retrieved from the database is completed, the FDS publishes the validation result to an exchange hosted created the message broker
- The message consumers consume the message from the exchange and react accordingly[14]

4.4.3. Managing Validation Rules

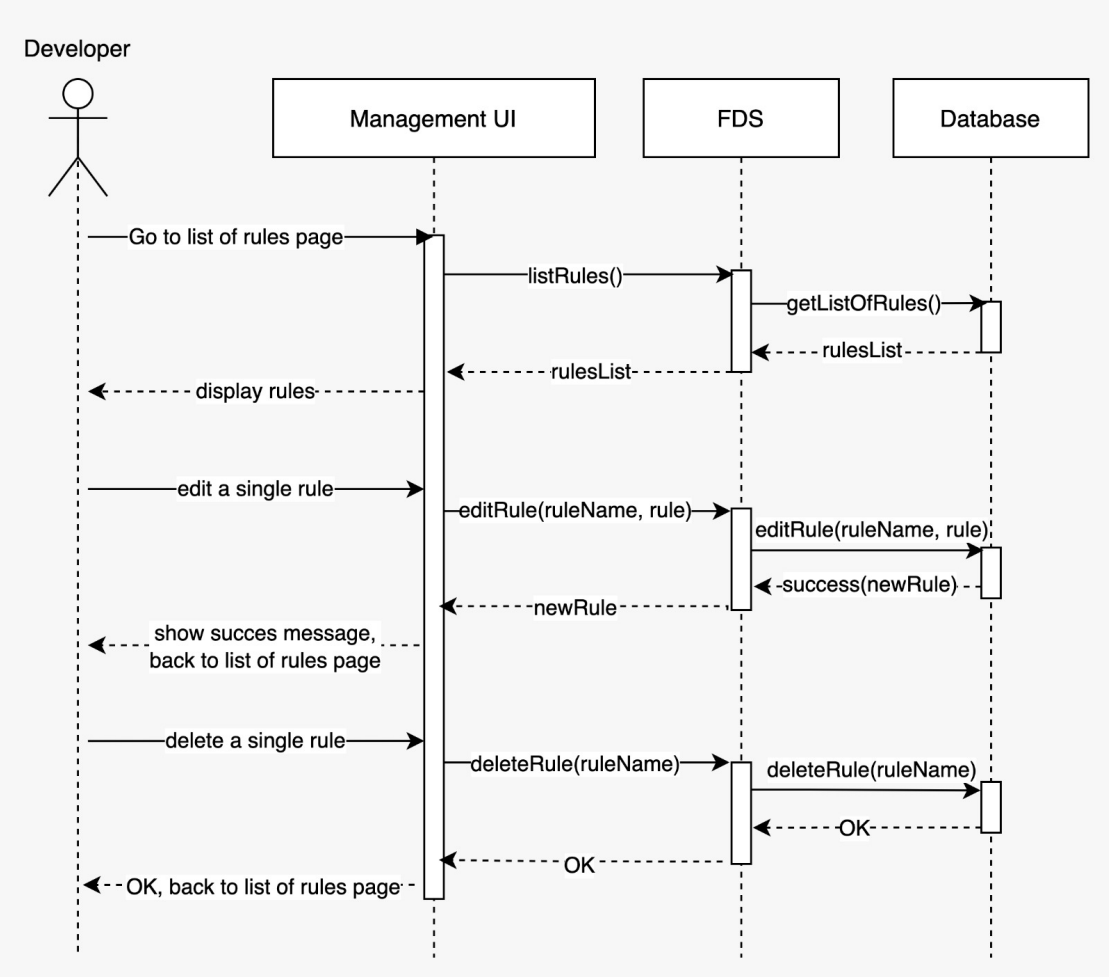

Figure 4.6.: *System sequence diagram for validation rules management*

Another sequence can also be defined as a result of an analysis of the following use case:

"As an employee, I want to manage my own rule to validate users, so

[14]For example: sending an email notification if the fraud score exceeds 0.7.

that I can use my expertise to find suspicious customers as efficiently as possible without the communication overhead with other teams"

The following sequence will be executed by the system as a detailed flow for a validation rule management:

- A user (e.g. Developer) can access the management UI and go to the page that displays a list of available validation rules
- The FDS retrieves a list of validation rules from the database
- User can click on a single rule and edit the rule
- The management UI makes an HTTP PUT[15] request to the database to edit an existing rule
- The FDS receives the HTTP request, modify the rule on the database and returns the edited rule as a response
- The management UI displays a success message and redirects user back to the list of rules page
- User can click on a single rule and delete the rule
- The management UI makes an HTTP DELETE[16] request to the database to delete an existing rule
- The FDS receives the HTTP request and delete the rule on the database, returning a 204[17] status code as an identifier of a successful operation
- The management UI displays a success message and redirects user back to the list of rules page

4.5. Software Design

The system was implemented using the Model-View-Controller (MVC) programming approach. Krasner and Pope [8] discussed the benefit of using the MVC pattern to provide modularity by isolating functional components of the system, making it easier to design and modify.

4.5.1. Model

As mentioned by Krasner and Pope [8], an application's model is a domain specific simulation or implementation of a structure. The model component of an MVC application contains business logic and manages the state of an application as well as its storage. The essential models of the system can be defined by revisiting the sequences listed on section 4.4 and identifying the important structures needed to fulfill the requirements needed.

[15] In [2, "9.6 PUT"], HTTP PUT method is described as a method to store or modify an entity, defined by the Request-URI

[16] In [2, "9.7 DELETE"], HTTP DELETE method is described as a method to delete a resource on the host server, pointed by the Request-URI

[17] In [2, "10.2.5 204 No Content"], the 204 status code should be used if the server fulfilled the request, but no data should be returned by the HTTP response

Validation Rule

A validation rule is a structure of information used in a validation process by supplying the FDS the necessary information to make an HTTP request to an external endpoint and evaluating its response, affecting the overall fraud score of a validation process through its evaluation result. Through a detailed analysis of the sequence listed on subsection 4.4.2, it is essential that the *validation rule* model contains the following attributes:

- A URL pointing to an external endpoint
- A list of conditions to evaluate the response returned by the external endpoint
- A unique identifier
- A fail score, which determine the severity of the rule if the evaluation failed

It might also be necessary to have an identifier in the validation rule to skip its evaluation in certain cases. Other than that, as the FDS would make an HTTP request to an external endpoint based on the information listed on a validation rule, the following attributes are needed to provide a more robust configuration:

- HTTP method to be used to make the request
- Request header[18]
- Request body

As the FDS interacts with external endpoints, there is no guarantee that the external endpoint will always be accessible. An additional attribute to specify and configure a retry strategy in such cases can be useful. However, a retry strategy can be really specific to its implementation and therefore will be discussed in **TODO: Add retry strategy implementation**.

An additional *priority* attribute is also provided to enable the possibility to run rule validations according to its priority order.

The *condition* attribute plays an important role for a validation rule, as it defines how the response returned by the external endpoint should be to pass a rule evaluation. It is intended to design the condition attribute to be robust and configurable. The *path* of a condition defines a JSONPath[4] expression to access information available of the current validation scope, such as customer information or response returned by the external endpoint. The *type* attribute of a condition determines the type of the attribute accessed by the *path* attribute. The *type* attribute determines which type of operators are available to use[19]. The *operator* attribute refers to a name of operator to be used to evaluate the condition (for example: "eq", "incl"). The available operator names are predefined and restricted to the condition's type attribute. More information regarding operators will be discussed in **TODO: Add retry strategy implementation**. The *failMessage* attribute of a condition refers to a message that is going to be appended to the validation result's *messages* attribute after a validation is completed.

[18] In [2, "5.3 Request Header Fields"]: request header is defined as additional information passed by the client to the host server about the particular request or about the client.

[19] For example: a condition with *type* "string" cannot use the "incl" *operator*, because the "incl" *operator* is only available for "array" *type*

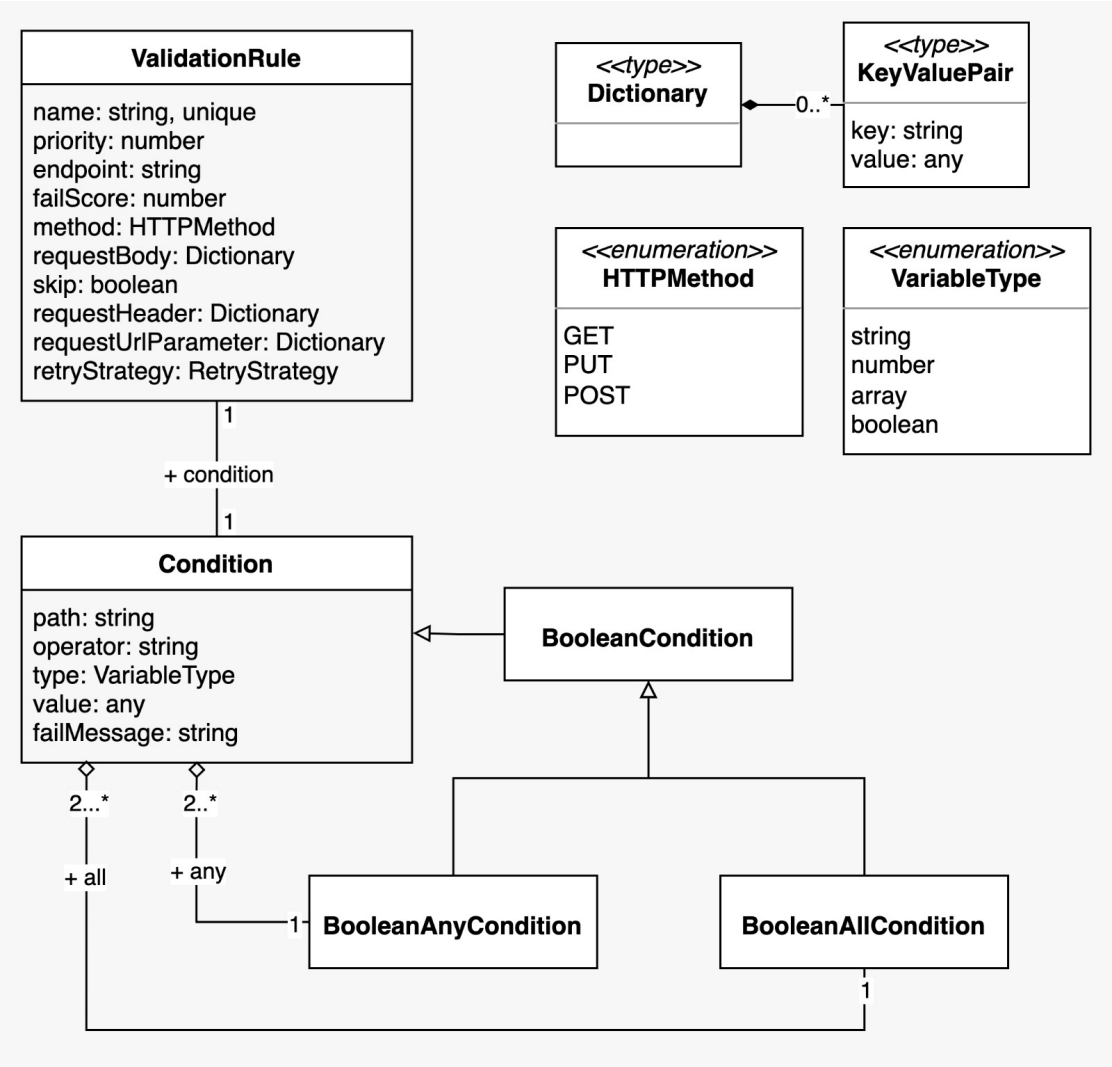

Figure 4.7.: *UML diagram of the validation rule model*

```
{
  "name": "Example",
  "skip": false,
  "priority": 2,
  "endpoint": "http://localhost:8000/validate",
  "method": "GET",
  "failScore": 0.7,
  "condition": {
    "path": "$.response.statusCode",
    "type": "number",
    "operator": "eq",
    "value": 200,
    "failMessage": "Status code doesn't equal to 200"
  },
  "requestUrlParameter": {},
  "requestBody": {},
  "requestHeader": {}
```

```
}
```

Listing 4.1: *Validation rule example (JSON)*

A validation rule might contain more than a single condition to pass an evaluation. In such cases, the user need to define whether how to determine whether an evaluation should pass: either pass an evaluation if **ALL** the conditions is true or pass an evaluation if **AT LEAST ONE (ANY)** of the conditions is true. This can be achieved by having the *condition* attribute as an object with a single attribute, either *all* or *any* and an array of conditions as the attribute's value.

```
{
  "name": "Example",
  "skip": false,
  "priority": 2,
  "endpoint": "http://localhost:8000/validate",
  "method": "GET",
  "failScore": 0.7,
  "condition": {
    "all": [
      {
        "path": "$.response.statusCode",
        "type": "number",
        "operator": "eq",
        "value": 200,
        "failMessage": "Status code doesn't equal to 200"
      },
      {
        "path": "$.response.body.valid_address",
        "type": "boolean",
        "operator": "eq",
        "value": false,
        "failMessage": "Address is invalid"
      }
    ]
  },
  "requestUrlParameter": {},
  "requestBody": {},
  "requestHeader": {}
}
```

Listing 4.2: *Validation rule example with ALL conditions (JSON)*

```
{
  "name": "Example",
  "skip": false,
  "priority": 2,
  "endpoint": "http://localhost:8000/validate",
  "method": "GET",
  "failScore": 0.7,
  "condition": {
    "any": [
      {
        "path": "$.response.statusCode",
        "type": "number",
        "operator": "eq",
```

```
14          "value": 200,
15          "failMessage": "Status code doesn't equal to 200"
16        },
17        {
18          "path": "$.response.body.valid_address",
19          "type": "boolean",
20          "operator": "eq",
21          "value": false,
22          "failMessage": "Address is invalid"
23        }
24      ]
25    },
26    "requestUrlParameter": {},
27    "requestBody": {},
28    "requestHeader": {}
29  }
```

Listing 4.3: *Validation rule example with ANY conditions (JSON)*

Validation Result

A validation result is the result of a validation process. A validation result contains a resulting fraud score, which is the probability of a certain customer being a fraud. The algorithm to calculate the fraud score will be discussed as part of the **TODO: Link implementation** By evaluating the sequences listed on subsection 4.4.1 and subsection 4.4.2, the following attributes are needed:

- A unique validation ID
- A fraud score

Furthermore, information regarding the total checks, runned checks, and additional information that contains the start and end date of the validation process as well as the customer information used for the validation are essential. The customer information is generic, and the system should be able to do a validation process regardless of its structure. The validation result should also return a list of validation rule names, whose evaluations are skipped due to its *skip* attribute being set to *true*.

Even though the resulting fraud score determines the probability of a customer being a fraud, it is also important to further analyze the actual evaluation result of each validation rule. For example, a certain action can be run if the evaluation of a specific rule failed or the information regarding the rule evaluations can also be used to display the current progress of a validation process (implementation of this specific functionality will be discussed more in **TODO: cite to implementation**).

To provide such information on a validation result, the *events* attribute will be introduced, which refers to rule evaluation events within a validation process. A rule evaluation event should contain the following attributes:

- Unique identifier of the event (name of the rule being evaluated)
- Status of the event (not started, failed, passed or running)
- If available, start date of a rule evaluation event
- If available, end date of a rule evaluation event

- A list of messages, containing error messages of an evaluation event

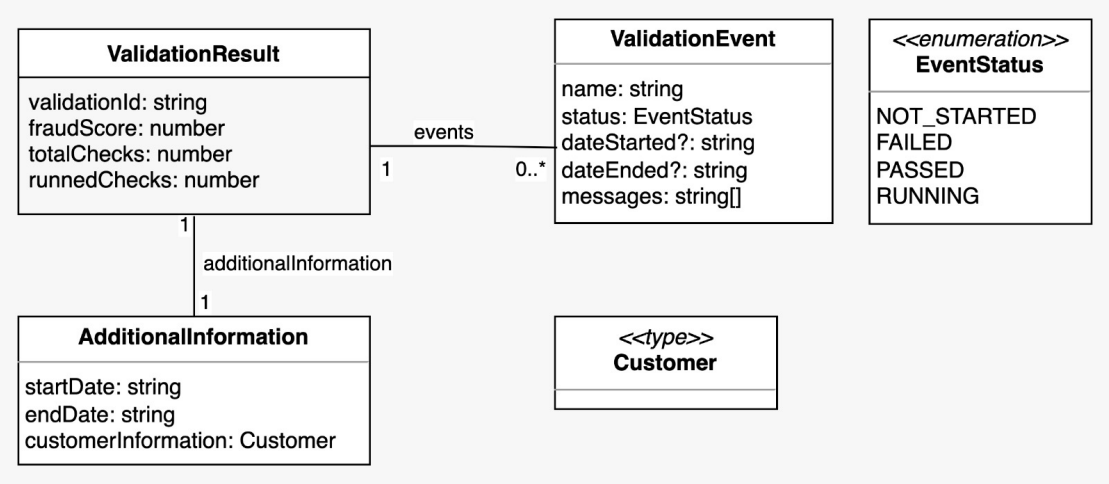

Figure 4.8.: *UML diagram of the validation result model*

```
{
  "validationId": "3112dc4a-45f5-41b8-883a-715dcbe9a490",
  "totalChecks": 3,
  "runnedChecks": 2,
  "skippedChecks": [
    "Skip rule",
  ],
  "additionalInfo": {
    "startDate": "2022-06-12T09:16:24.618Z",
    "customerInformation": {
      "firstName": "Scooby",
      "lastName": "Doo",
      "address": {
        "streetName": "Suite 5000 185 Berry St",
        "city": "San Fransisco",
        "state": "CA",
        "country": "United States",
        "postalCode": 94107
      },
      "email": "scooby-doo@fraud.co",
      "phoneNumber": "123131123"
    }
  },
  "events": [
    {
      "name": "User's email domain is not blacklisted",
      "status": "PASSED",
      "dateStarted": "2022-06-12T09:16:34.694Z",
      "dateEnded": "2022-06-12T09:16:39.725Z",
      "messages": []
    },
    {
      "name": "User's email is not blacklisted",
      "status": "FAILED",
```

```
35        "dateStarted": "2022-06-12T09:16:39.725Z",
36        "dateEnded": "2022-06-12T09:16:44.756Z",
37        "messages": [
38          "User's email is blacklisted!"
39        ]
40      }
41    ],
42    "fraudScore": 0.425
43  }
```

Listing 4.4: *Validation result example (JSON)*

4.5.2. View

4.5.3. Controller

5. Implementation

[Beschreibung der Implementierung[1]auf Basis des Entwurfs und der Methodologie / der geplanten Vorgehensweise zur Problemlösung im Kontext der Anforderungen. Hier ist Raum für Listings, wie z.B. das nun Folgende:

```
1 const helloWorld = "Hello, world!"
2 console.log(helloWorld)
```

Listing 5.1: *Ein Beispiel: Hello World (JavaScript)*

Umfangreicher Quell-Code sollte in den Anhang ausgelagert werden.]

5.1. Model

5.2. View

5.3. Controller

Optionals:

- Architecture -> (can be in controller)
- Techs

[1]Beachten Sie bei der Implementierung und deren Dokumentation bitte Clean Code Empfehlungen (vgl. hierzu z.B. [11]).

6. Test

[Beschreibung, wie Sie auf Basis des geplanten Testvorgehens was mit welchen Kriterien und Technologien getestet haben]

7. Demonstration and Evaluation

[Beschreibung der Ergebnisse aus allen voran gegangenen Kapiteln sowie der zuvor generierten Ergebnisartefakte mit Bewertung, wie diese einzuordnen sind]

7.1. Ausblick

[Beschreibung und Begründung potenzieller zukünftiger Folgeaktivitäten im Zusammenhang mit Ihrer Arbeit (z.B. weitere Anforderungen, Theoriebildung, ...]

Bibliography

[1] Helmut Balzert, Marion Schröder, and Christian Schaefer. *Wissenschaftliches Arbeiten. Ethik, Inhalt & Form wiss. Arbeiten, Handwerkszeug, Quellen, Projektmanagement, Präsentation*. 2. Auflage. Herdecke, Witten: W3L, 2011. ISBN: 978-3-86834-034-1.

[2] R. Fielding et al. *RFC 2616, Hypertext Transfer Protocol – HTTP/1.1*. Tech. rep. 1999. URL: http://www.rfc.net/rfc2616.html.

[3] Norbert Franck and Joachim Stary. *Die Technik wissenschaftlichen Arbeitens: eine praktische Anleitung*. 17. Auflage. Paderborn: Schöningh, 2013. ISBN: 978-3-50697-027-5.

[4] Jeff Friesen. "Extracting JSON Values with JsonPath". In: *Java XML and JSON: Document Processing for Java SE*. Berkeley, CA: Apress, 2019, pp. 299–322. ISBN: 978-1-4842-4330-5. DOI: 10.1007/978-1-4842-4330-5_10. URL: https://doi.org/10.1007/978-1-4842-4330-5_10.

[5] Erich Gamma et al. *Design Patterns*. Addison-Wesley, 1995.

[6] David Garlan. "Software architecture". In: *Proceedings of the conference on The future of Software engineering - ICSE '00* (2000). DOI: 10.1145/336512.336537.

[7] ISO. *ISO/IEC 25010:2011*. Mar. 2011. URL: https://www.iso.org/standard/35733.html.

[8] Glenn Krasner and Stephen Pope. "A cookbook for using the model-view controller user interface paradigm in Smalltalk-80". In: *Journal of Object-oriented Programming - JOOP* 1 (Jan. 1988).

[9] Loriot. *Möpse und Menschen. Eine Art Biographie. Zurich*. In: faz.net. Online: https://media0.faz.net/ppmedia/aktuell/feuilleton/1461387463/1.721778/format_top1_breit/die-steinlaus-trotzt-seit.jpg; letzter Zugriff: 13 VI 19. 1983.

[10] Loriot. *Steinlaus, Loriot Katalog, Diogenes Verlag, Zürich*. In: tagblatt.de. Online: https://www.tagblatt.de/Bilder/Loriots-legendaere-Steinlaus-Loriot-Katalog-1993-2003-125217h.jpg; letzter Zugriff: 14 VI 19. 1993, 2003.

[11] Robert Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*. 1st ed. Pearson, 2008.

[12] Alexey Melnikov and Ian Fette. *The WebSocket Protocol*. RFC 6455. http://www.rfc-editor.org/rfc/rfc6455.txt. RFC Editor, 2011. URL: http://www.rfc-editor.org/rfc/rfc6455.txt.

[13] Pschyrembel online. *Steinlaus*. Online: https://www.pschyrembel.de/Steinlaus/K0LHT; letzter Zugriff: 14 VI 19. 2016.

[14] Alan Jay Smith. "Cache Memories". In: *ACM Computing Surveys* 14.3 (1982), pp. 473–530. DOI: 10.1145/356887.356892.

[15] Wikipedia. *Academic Use*. Online: https://en.wikipedia.org/wiki/Wikipedia:Academic_use; letzter Zugriff: 13 VI 19. 2019.

8. List of Abbreviations

9. Glossary

A. Appendix

A.1. Quell-Code

A.2. Tipps zum Schreiben Ihrer Abschlussarbeit

- Achten Sie auf eine neutrale, fachliche Sprache. Keine „Ich“-Form.
- Zitieren Sie zitierfähige und -würdige Quellen (z.B. wissenschaftliche Artikel und Fachbücher; nach Möglichkeit keine Blogs und keinesfalls Wikipedia[1]).
- Zitieren Sie korrekt und homogen.
- Verwenden Sie keine Fußnoten für die Literaturangaben.
- Recherchieren Sie ausführlich den Stand der Wissenschaft und Technik.
- Achten Sie auf die Qualität der Ausarbeitung (z.B. auf Rechtschreibung).
- Informieren Sie sich ggf. vorab darüber, wie man wissenschaftlich arbeitet bzw. schreibt:
  - Mittels Fachliteratur[2], oder
  - Beim Lernzentrum[3].
- Nutzen Sie LaTeX[4].

[1] Wikipedia selbst empfiehlt, von der Zitation von Wikipedia-Inhalten im akademischen Umfeld Abstand zu nehmen [15].

[2] Z.B. [1], [3]

[3] Weitere Informationen zum Schreibcoaching finden sich hier: `https://www.htw-berlin.de/studium/lernzentrum/studierende/schreibcoaching/`; letzter Zugriff: 13 VI 19.

[4] Kein Support bei Installation, Nutzung und Anpassung allfälliger LaTeX-Templates!

Eidesstattliche Versicherung

Hiermit versichere ich an Eides statt durch meine Unterschrift, dass ich die vorstehende Arbeit selbstständig und ohne fremde Hilfe angefertigt und alle Stellen, die ich wörtlich oder annähernd wörtlich aus Veröffentlichungen entnommen habe, als solche kenntlich gemacht habe, mich auch keiner anderen als der angegebenen Literatur oder sonstiger Hilfsmittel bedient habe. Die Arbeit hat in dieser oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen.

_____

Datum, Ort, Unterschrift