Designing Microservice-Oriented Application Frameworks

Yunhyeok Lee
Dept. of Computer & Information Science
University of Massachusetts Dartmouth
Dartmouth, Massachusetts, U.S.A.
ylee7@umassd.edu

Yi Liu
Dept. of Computer & Information Science
University of Massachusetts Dartmouth
Dartmouth, Massachusetts, U.S.A.
yliu11@umassd.edu

***Abstract*—An application framework is a reusable "skeleton" application that can be specialized to develop custom applications. As microservices become more popular in the software industry, the importance of methodologies for designing microservices-oriented frameworks is increasing. However, the existing methodologies for designing application frameworks, which were developed for the monolithic environment, do not fit the microservice architecture well. To address this issue, this study extends Schmid's systematic generalization approach to enable the design of microservice-oriented application frameworks. The methodology introduces communication spots, which define the execution orders among services, and communication styles, which define the communication relationship among services, to capture the communication characteristics in microservice environments. The methodology also proposes the strategies in microservice generalization and the usage of integration patterns in transforming a microservice-oriented application into a microservice-oriented framework. The new methodology can facilitate a developer to systematically design a microservice-oriented application framework by generalizing from a microservice application. A case study is used to demonstrate how to apply the proposed methodology to design a microservice fraud-detection framework.**

***Keywords*—software frameworks, microservice-oriented, design methodology**

I. Introduction

An application framework is "a generic application that allows different applications to be created from a family of applications." [18] Application frameworks facilitate the developers to reuse the framework's software artifacts to efficiently construct customized applications in particular business domains. An application framework contains the common aspects (*frozen spots*) that all family members have and variable aspects (*hot spots*) that vary among the application in the family [18]. The *hot spots* allow the framework to be customized to a specific member of the family.

The design of an application framework can be much more complicated than the design of a single application because the framework needs to address both common aspects and variable aspects in the application domain. Thus, the framework design requires a systematic approach.

Schmid[18] proposed a methodology of designing application frameworks using systematic generalization. The methodology examines the design of an existing application, identifies the frozen spots and hot spots in the family, and generalizes the application structure to construct a framework [6]. Built on the assumption of using the monolithic environment (single code-base and one-time deployment), the methodology is well suitable for designing object-oriented monolithic frameworks in which the components are designed to communicate based on the class relationships, such as inheritance, polymorphism, and composition. Microservice architecture [8] distinguishes itself from monolithic architectures by characteristics, such as modularity, fine grained services and communication mechanisms. In a microservice-oriented application, each service has its own process, which serves a single purpose and communicates with other services through lightweight mechanisms, such as APIs. As microservices are increasingly used in the development world, microservice-oriented application frameworks will be beneficial for the development of microservice-oriented applications in the particular business domains that the frameworks are constructed for. However, few study has been published on the methodology of designing microservice-oriented application frameworks. Schmid's approach is applicable for generalizing the hot spots and frozen spots within a microservice, but it is not sufficient to address the inter-microservices communications.

This research aims to propose a methodology for designing microservice-oriented application frameworks. The proposed approach is built on Schmid's systematic generalization methodology and extends it to fit into the microservices environment.

The rest of the paper is organized as follows. Section II briefly introduces the microservice architecture, Schmid's application framework design methodology and microservices integration patterns. Section III presents a methodology for designing microservice-oriented application frameworks. Section IV uses a case study to present how to apply the proposed methodology in designing a microservice fraud detection framework. Section V discusses related works similar to our approach, and Section VI concludes the work.

II. Background

This section provides a brief introduction to the microservices architecture and microservices integration patterns as

DOI reference number: 10.18293/SEKE2022-163

well as an overview of Schmid's systematic framework design from which our study extends.

A. Microservices Architecture

Martin Fowler defined the term "Microservices" [8] as a method to build a software application with a set of small services. Each service has its own process, which serves a single purpose and communicates with other services through application programming interfaces (API) [8]. *Modularity, fine-grained services,* and *simple communication mechanism* are the three key characteristics of microservices.

A microservice-oriented application contains a suite of independent modules in a system. Each of thoese modules, also called a microservice, encapsulates its domain logic and contributes to the overall functionality of its system, unlike the monolith which puts all the functionalities in a single process [8]. This modularity improves the flexibility of the development and the deployment of each service and increases the overall comprehensibility of the system.

The size of each microservice is comparatively small. Each service should focus on a single business capability to support low coupling within the system. The independent fine-grained services allow for a low cost of system maintenance and evolution into the future.

The microservices architecture focuses on lightweight communication mechanisms instead of hiding complexities in the communications. The HTTP request-response with resource APIs and lightweight messaging are commonly used in microservices to provide "dumb pipes" [8]. A simple communication approach enables changes in services without modifying the central service communication bus and offers the system better scalability to the overall system [3].

B. Microservice Integration Patterns

Microservices must be properly composed for any given system to function. There could be a large number of services in a system. In addition, a service may be reused within different scopes, which will increase the complexity of that service's composition. Service integration is crucial and challenging in the field of microservices architecture.

The case study of this research adopts the general service integration patterns [19], which are based on the interservice communication mechanisms within microservices. These patterns are *Synchronous Messaging*, *Asynchronous Messaging*, and *Hybrid Messaging*.

1) Synchronous Messaging Pattern: The communication between microservices is synchronous if microservice A sends microservice B a request that requires a response and A is blocked while waiting for said response [16]. The *Synchronous Messaging* pattern is designed to integrate microservices when they communicate via synchronous messaging.

2) Asynchronous Messaging Pattern: The communication is asynchronous if microservice A sends a request but is not blocked while waiting for a response if there is any [16]. The *Asynchronous Messaging* pattern is designed to integrate microservices that communicate via asynchronous messaging.

3) Hybrid Messaging Pattern: To realize a business need, a combination of synchronous and asynchronous messaging is typically required within a system. The *Hybrid Messaging* pattern provides a solution by combining aspects of the *Synchronous Messaging* pattern and the *Asynchronous Messaging* pattern.

C. Systematic Framework Design

Schmid's methodology of the systematic construction of an application framework consists of four steps [18]:

1) Creation of a Fixed Application Model: Schmid's approach starts by constructing a fixed application model with an object-oriented design for a specific application within the family.

2) Hot Spot Analysis and Specification: Once a complete model exists, the framework designer analyzes the model and domain to discover and specify potential hot spots.

3) Hot Spot High-Level Design: The features of any hot spots are accessed through the common interface of the abstract class. However, the design of the hot spot subsystem enables different concrete subclasses of the base class to be used to provide variant behaviors.

4) Generalization Transformation: The approach seeks to generalize the design around these hot spots by applying systematic transformations of the design that are driven by the analysis of the hot spot.

III. Methodology

The proposed methodology uses Schmid's approach as a basis and extends it to fit into the microservice-oriented environment. The methodology consists of six steps as described below.

A. Step 1. Develop a Solid Microservice-Oriented Application

Just like Schmid's approach, the proposed methodology starts with the construction of a representative application in the framework family. However, this application should be microservice-oriented, unlike the object-oriented, monolithic application in Schmid's approach.

B. Step 2. Identify Hot Spots and Frozen Spots

Aspects that vary among application family members are hot spots [18]. The different implementations to the hot spots result in different applications within the family. A framework is customized to a specific application with the specific implementations to the hot spots. The aspects that are common to all the application family members are called frozen spots [18]. These frozen spots are the basis of designing the overall structure of the framework and are fixed and reusable for all the applications within the family.

When designing a microservice-oriented framework, any hot spots and frozen spots should be identified for each service. Commonality and variability analysis approaches [5, 12] are beneficial for identifying the hot spots and frozen spots.

C. Step 3. Analyze and Specify Hot Spots

We adopt the Schmid's approach in analyzing the high-level hot spots and specifying the details [18]. All identified hot spots from the domain are collected and evaluated. Each hot spot is specified by a short description of its purpose, its common responsibility, the kinds of variability and the multiplicity.

D. Step 4. Design Hot Spot Subsystems

A hot spot is captured within the scope of a microservice and implemented by a hot spot subsystem. Schmid's strategy of designing hot spot subsystems is adopted for this phase. A hot spot subsystem consists of an (abstract) interface defining the common responsibilities, concrete implementations (for addressing variability) to the interface, and additional classes and relationships [18]. Design patterns [17] are beneficial for determining the structures of the hot spot subsystems on capturing the abstract interface and concrete implementations and their relations.

E. Step 5. Identify Communication Spots and Communication Styles

In a monolithic object-oriented application, communications among the components are typically conducted through the procedure calls via inheritance, polymorphism, or composition. Such communication is applicable for the components within a microservice, but would not work between the microservices.

We use *communication spots* and *communication styles* to specify how the microservices communicate within a system. We define a *communication spot* as the execution order between two services. For example, microservice A executes prior to microservice B.

A *communication style* is defined as the procedure between the server and client to communicate when there is a request from the client side and the server side is expected to respond to the client's request. We use communication styles to describe the communications between microserivces or between clients and microservices.

The communication styles are categorized as synchronous and asynchronous, where the synchronous method is that a client side sends a request to a server-side service and waits for its response, and the synchronous method is that a client-side keeps sending requests to the server-side service without waiting for the acknowledgment of the previous response [11].

We use the notation $A(C, S)$ to describe that client-side C communicates with server-side service S through the asynchronous style. We use $S(C, S)$ to describe that client-side C communicates with server-side service S through the synchronous style.

This stage produces two descriptions: the *communication spot description* that consists of a list of the execution order of each pair of services, and the *communication style description* that contains a list of each pair of client-side and server-side service's communication style.

F. Step 6. Transform into a Microservice-Oriented Framework

Schmid's generalization transformation strategy uses the object-oriented techniques, such as inheritance, polymorphism, and composition, to generalize the design around the hot spots for constructing the final framework structure. Such transformation strategy is applicable within each microservice. Beyond the structure of each service, a microservice-oriented framework also needs to reflect the microservice generalization and service communications.

We propose two options in generalizing the microservices. The first option, as shown in Fig. 1, captures the abstract interface and concrete implementations to the interface within a microservice; while the second option, as shown in Fig. 2, captures the abstract microservices in the framework only.

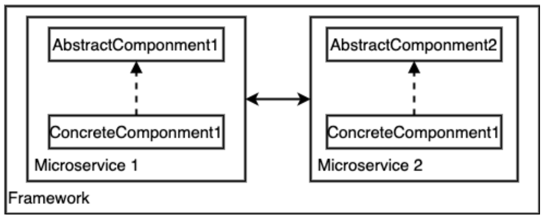


Fig. 1. Microservice Generalization - Option 1

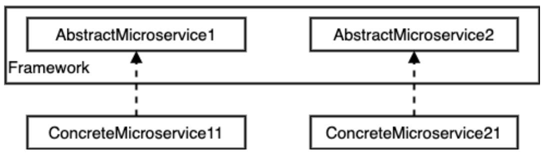


Fig. 2. Microservice Generalization - Option 2

The microservices should be integrated properly in the framework. The identified communication spots and communication styles determine the final integration transformation. The three integration patterns (Synchronous Messaging, Asynchronous Messaging, and Hybrid Messaging), described in Section II.B, can facilitate the integration process.

IV. Case Study

In this section, the design of a fraud detection application framework is used to illustrate how the proposed approach can be applied.

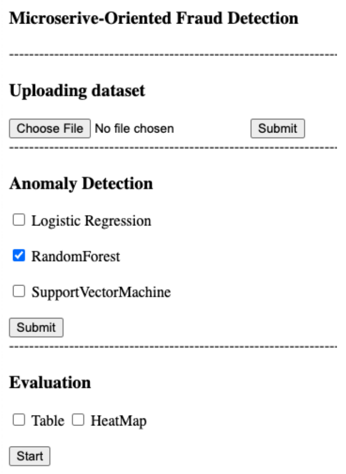
A. Overview of a Fraud Detection Application

Inspired by the fraud detection analysis on fraudulent credit card transactions [15], we have developed a prototype microservice application that uses machine-learning techniques to predict fraudulent credit card transactions. The application consists of four microservices, which are described below:

- *Dataset Uploading service* is for users to upload a dataset to the application. The CSV file format is accepted by default; however, the service converts a non-CSV files (such as XLSX) to CSV format.
- *Preprocessing service* normalizes data and selects important features for further analysis.

- *Fraud Detection service* applies the machine learning algorithms, such as Random Forest [21], Support Vector Machine [22], and Logistic Regression [4], etc., to detect fraud cases in the dataset.
- *Evaluation service* presents the results from *Fraud detection* as tables and heatmaps.

Figure 3(a) shows the user interface of the prototype application, while 3(b) and (c) illustrate the results by applying the *Random Forest* algorithm in a heatmap and a table.



Microserive-Oriented Fraud Detection

Uploading dataset

Choose File No file chosen Submit

Anomaly Detection

☐ Logistic Regression

☑ RandomForest

☐ SupportVectorMachine

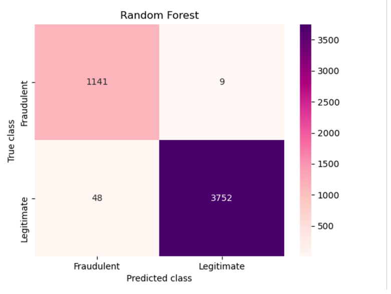
Submit

Evaluation

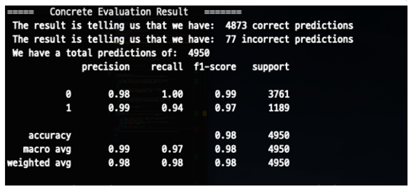
☐ Table ☐ HeatMap

Start

(a)



(b)



```
=====   Concrete Evaluation Result   =======
 The result is telling us that we have:  4873 correct predictions
 The result is telling us that we have:  77 incorrect predictions
 We have a total predictions of:  4950
              precision    recall  f1-score   support

           0       0.98      1.00      0.99      3761
           1       0.99      0.94      0.97      1189

    accuracy                           0.98      4950
   macro avg       0.99      0.97      0.98      4950
weighted avg       0.98      0.98      0.98      4950
```

(c)

Fig. 3. Prototype Fraud Detection System

In this research, the scope of the fraud detection application family is constrained to analyze only stationary datasets on classical computers using CPUs.

B. Design of Fraud Detection Framework

In order to allow the developers to design their own user interfaces (UIs), the fraud detection framework does not include UI components, but provides interfaces to connect to UIs.

Since a specific microservice application has already been built, the design of the fraud detection framework begins with step 2 of the design approach.

1) Identification of Frozen Spots and Hot Spots: By analyzing the design decisions and the scope of the application family, we choose the following frozen spots:

- *Frozen Spot 1.* The data input and analysis order is fixed, that it, dataset upload, data preprocessing, fraud analysis and detection, and presentation of results.
- *Frozen Spot 2.* A dataset is uploaded one at a time and then converted to CSV format.
- *Frozen Spot 3.* Machine learning algorithms are used in the *Fraud Detection* service.
- *Frozen Spot 4.* Users should be able to upload a dataset, choose machine learning algorithms and view the results.

After examining the scope and the prototype application, we identify the following aspects that vary among applications in the fraud detection family, thus define them as hot spots:

- *Hot Spot 1.* Variability in the file formats of datasets uploaded by users in the *Dataset Uploading* service. In addition to the default CSV format, other file formats should also be supported, such as XLXS, mat, txt, etc.
- *Hot Spot 2.* Variability in the data normalization techniques in the *Preprocessing service*.
  Data normalization can be done using various methods, such as decimal scaling, min-max normalization, etc. Some text-based datasets may need natural language processing [13] before further analysis.
- *Hot Spot 3.* Variability in the feature selection techniques in the *Preprocessing* service.
  Various methods can be applied to select the important features such as Recursive Feature Elimination [2], LASSO [7], etc.
- *Hot Spot 4.* Variability in the machine learning algorithms applied to analyze a dataset in *Fraud Detection* service.
  Various machine learning algorithms is applied to analyze a dataset. These algorithms include supervised learning (e.g. Random Forest) and semi-supervised learning (e.g. Convolutional Neural Network [1]).
- *Hot Spot 5.* Variability in the result presentation in the *Evaluation* service.
  In addition to the heatmap and table presentation implemented in the prototype, other data visualization techniques can be added, such as AUROC curve [14].

2) Hot Spot High-Level Specification and Design: This stage of the design involves the detailed specification of each identified hot spot and the design of the hot spot subsystems.

Due to the page limit of this paper, we only present the specification of hot spot 4, and its hot spot subsystem design.

Hot Spot 4: Variability in the machine learning algorithms applied to analyze a dataset in *Fraud Detection* service.

- Description: To allow a variety of machine learning algorithms.
- Common responsibility is to perform fraud detection on a dataset.
- The kinds of variability required are the algorithms in the categories of unsupervised learning, supervised learning, and semi-supervised learning.

- The multiplicity is one since one machine learning algorithm is used at a time.

The various machine learning algorithms are considered to be interchangeable in the fraud detection. The subsystem of hot spot 4 should allow new algorithms to be added and existing algorithms to be modified or removed without impacting the remaining parts of the system. In addition, a convenient way to access each algorithm should be provided. The *Strategy* design pattern defines "a family of algorithms, encapsulates each one, and makes them interchangeable; Strategy lets the algorithm vary independently from clients that use it." [9] This pattern is the best fit for organizing and managing the independent machine learning algorithms.

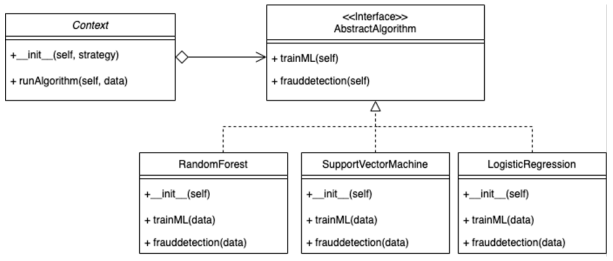


Fig. 4. Hot Spot Subsystem of Fraud Detection

As illustrated in Fig. 4, the interface *AbstractAlgorithm* defines the methods that are common in the machine learning algorithms which are supported in this application family; A machine learning algorithm, such as *Random Forest, Support Vector Machine, Logistic Regression*, etc., is implemented as a concrete class to the *AbstractAlgorithm* interface; The *Context* maintains a reference to each machine learning algorithm object and forwards the requests from its clients to the machine learning algorithm objects.

3) Identification of Communication Spots and Communication Styles: The execution order of a pair of services that communicate with one another is identified as a communication spot. By examining the framework's scope, we can capture the following communication spots:

- *Communication Spot 1*: The *Dataset Uploading* service executes prior to the *Preprocessing* service.
- *Communication Spot 2*: The *Preprocessing* service executes prior to the *Fraud Detection* service.
- *Communication Spot 3*: The *Fraud Detection* service executes prior to the *Evaluation* service.

The *Dataset Uploading* service communicates with a client through the synchronous communication method. A client sends a synchronous request to the *Dataset Uploading* service.

We analyze the communication styles by inspecting how each serve-side service responds to a client-side's request. The identified communication styles are summarized below:

- S(client, Dataset Uploading): A client sends a synchronous request to the *Dataset Uploading* service. Only a single file can be uploaded at a time.
- S(client, Preprocessing): A client sends a synchronous request to the *Preprocessing* service. One a single dataset is preprocessed at a time.
- A(Preprocessing, Fraud Detection): A client-side *Preprocessing* sends asynchronous requests to server-side *Fraud Detection* service. Multiple machine learning algorithms can be run concurrently.
- S(client, Evaluation): A client sends a synchronous request to *Evaluation* service. The result of one dataset is presented at a time.

4) Transformation to a Microservice-Oriented Framework: We adopt the first component organization approach (Fig. 1) to plug in the hot spot subsystems to the framework. For example, the Fraud Detection strategy (Fig. 4) along with its concrete implementations (*Random Forest, Support Vector Machine, Logistic Regression*, etc. are wrapped in the *Fraud Detection* service.

The identified communication styles indicate that the services use a mix of synchronous and asynchronous methods, thus, the *Hybrid Messaging* pattern is the best to be applied to integrate the services. With the help of the identified communication spots for the communication orders of the services, the high-level integration structure of the Fraud Detection framework is designed as shown in Fig. 5.

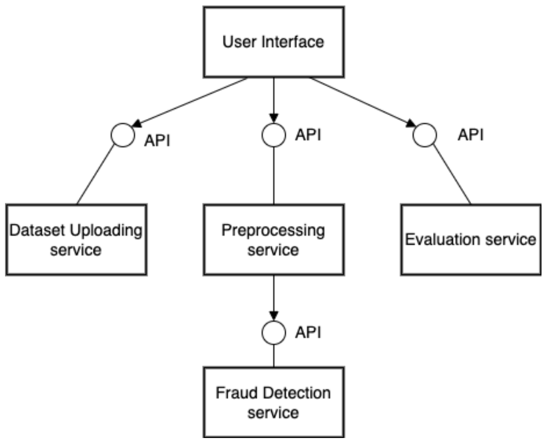


Fig. 5. Applying Hybrid Messaging Pattern to Integrate Services in Fraud Detection Framework

V. Discussion

Schmid's approach is object-oriented, generalizing the class structure of an application when designing application frameworks. There are also alternative systematic approaches that are not constrained to the object-oriented paradigm. Functional generalization is an example [6]. The function generalization approach generalizes the functional structure of an executable specification to produce an application framework. Hot spots are introduced to the design by replacing concrete operations with general abstract operations. These abstract operations become parameters of the generalized functions. Our proposed methodology can adopt such an alternative systematic approach by modifying step 4 and 6 to generalize functions instead of dealing with classes.

This study uses three integration patterns in the case study. There are also other alternative integration patterns that can be used in the transformation step. For example, Microsoft Azure proposed nine patterns [20] for integrating microservices in applications. Each pattern serves as a solution for a fine-grained integration problem. To construct a system with microservices, it is necessary to utilize most of these patterns simultaneously. Gupta [10] introduced six patterns to provide multiple integration approaches from varying perspectives.

VI. Conclusion

Microservice-oriented application frameworks will be beneficial for the development of microservice-oriented applications in the particular business domains that the frameworks are constructed for. The existing design approaches for application framework design are limited to a monolithic environment that does not fit into the microservice architecture. This research proposes a methodology that extends Schmid's systematic generalization methodology by introducing communication spots and transformation strategies that fit in the microservice architecture. A case study is used to demonstrate the usage of the proposed methodology in designing a microservice-oriented fraud detection framework. Future work will be focused on two directions. One direction is to enhance the fraud detection framework by introducing the support for time series, unsupervised learning algorithms, and the usage of quantum computing. The other direction is to examine the impact of the different computing environments (classical vs. quantum) on the framework design thus enhance the methodology. The enhancement from the first direction will set a practical basis for the study of the second direction.

References

[1] S. Albawi, T. A. Mohammed, and S. Al-Zawi. "Understanding of a Convolutional Neural Network". In: *2017 International Conference on Engineering and Technology (ICET)* (2017), pp. 1–6.

[2] Brandon D. Butcher and Brian J. Smith. "Feature Engineering and Selection: A Practical Approach for Predictive Models". In: *The American Statistician* 74 (2020), pp. 308–309.

[3] T. Cerný, M. J. Donahoo, and J. Pechanec. "Disambiguation and Comparison of SOA, Microservices and Self-Contained Systems". In: *Proceedings of the International Conference on Research in Adaptive and Convergent Systems* (2017).

[4] Wenlin Chen et al. "Density-Based Logistic Regression". In: *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining* (2013).

[5] James O. Coplien, Daniel Hoffman, and David M. Weiss. "Commonality and Variability in Software Engineering". In: *IEEE Softw.* 15 (1998), pp. 37–45.

[6] H. Conrad Cunningham, Yi Liu, and Pallavi Tadepalli. "Framework design using function generalization: a binary tree traversal case study". In: *ACM-SE 44*. 2006.

[7] Valeria Francesca Fonti and Eduard N. Belitser. "Feature Selection using LASSO". In: *VU Amsterdam Research Paper in Business Analytics* 30 (2017), pp. 1–25.

[8] M. Fowler. *Microservices: a Definition of This New Architectural Term*. 2014. URL: https://martinfowler.com/articles/microservices.html.

[9] Erich Gamma et al. "Design patterns: elements of reuseable object-oriented software". In: 1994.

[10] A. Gupta. *Microservice Design Patterns*. 2015. URL: https://uberconf.com/blog/arun_gupta/2015/04/microservice_design_patterns.

[11] Mingyu Lim. "Directly and Indirectly Synchronous Communication Mechanisms for Client-Server Systems Using Event-Based Asynchronous Communication Framework". In: *IEEE Access* 7 (2019), pp. 81969–81982.

[12] R AL-msie'Deen et al. "Detecting commonality and variability in use-case diagram variants". In: *ArXiv* abs/2203.00312 (2022).

[13] P. M. Nadkarni, L. Ohno-Machado, and W. W. Chapman. "Natural language processing: an introduction". In: *Journal of the American Medical Informatics Association : JAMIA* 18 (2011), pp. 544–51.

[14] S. Narkhede. *Understanding AUC - ROC Curve*. 2018. URL: https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5.

[15] R. Polantizer. *Fraud Detection in Python; Predict Fraudulent Credit Card Transactions*. 2021. URL: https://medium.com/@polanitzer/fraud-detection-in-python-predict-fraudulent-credit-card-transactions-73992335dd90.

[16] C. Richardson and F. Smith. *Microservices from design to deployment*. NGINX, Inc., 2016.

[17] H. A. Schmid. "Design Patterns for Constructing the Hot Spots of a Manufacturing Framework". In: *Journal Object Oriented Programming* 9 (1996), pp. 25–37.

[18] H. A. Schmid. "Systematic framework design by generalization". In: *Communication of ACM* 40 (1997), pp. 48–51.

[19] M. Wang. "Service Integration Design Patterns in Microservices". MA thesis. South Dakota State University, 2018.

[20] M. Wasson. *Microservices: a Definition of This New Architectural Term*. 2017. URL: https://azure.microsoft.com/en-us/blog/design-patterns-for-microservices/.

[21] Q. Wu et al. "ForesTexter: an Efficient Random Forest Algorithm for Imbalanced Text Categorization". In: *Knowledge-Based Systems* 67 (2014), pp. 105–116.

[22] Yue Zhang and Zhimeng Feng. "A SVM Method for Continuous Blood Pressure Estimation from a PPG Signal". In: *Proceedings of the 9th International Conference on Machine Learning and Computing* (2017).