



Technische Universität Berlin



# **Sparked, an intuitive user interface for the automated machine learning project CODA**

## **Bachelorarbeit**

am Fachgebiet Agententechnologien in betrieblichen Anwendungen und der  
Telekommunikation (AOT)

Prof. Dr.-Ing. habil. Sahin Albayrak  
Fakultät IV Elektrotechnik und Informatik  
Technische Universität Berlin

vorgelegt von

**Robin Maximilian Ruth**

Betreuer: Prof. Dr.-Ing. habil. Sahin Albayrak,  
Christian Geißler

Robin Maximilian Ruth  
Matrikelnummer: 316672  
Heidenfeldstr. 19  
10249 Berlin

# Erklärung der Urheberschaft

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit ohne Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form in keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Ort, Datum

Unterschrift

# Abstract

In this project, a web-based user interface for the machine learning software CODA is developed. From the basic requirements of creating a visually attractive web application used primarily for presentations that supports all the capabilities of an existing API, the full stack of software development will be shown from an empty page to the finished product.

# Contents

<b>Erklärung der Urheberschaft</b>	<b>II</b>
<b>Abstract</b>	<b>III</b>
<b>Contents</b>	<b>IV</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Basis</b>	<b>3</b>
2.1 State of the art . . . . .	4
<b>3 Problem analysis</b>	<b>8</b>
3.1 Requirements . . . . .	8
3.1.1 Nonfunctional Requirements . . . . .	9
3.1.2 Functional Requirements . . . . .	9
3.2 User Personas . . . . .	10
3.3 Summary . . . . .	11
<b>4 Concept</b>	<b>13</b>
4.1 Visual guidelines . . . . .	13
4.1.1 Looking from the users' point of view . . . . .	13
4.2 Architecture . . . . .	15
4.2.1 The 30000-foot view . . . . .	16
4.2.2 Workflows . . . . .	17
4.2.2.1 Workflow: Create and start an order . . . . .	17
4.2.2.2 Workflow: Listing orders . . . . .	19
4.2.2.3 Workflow: Evaluation . . . . .	20
<b>5 Implementation</b>	<b>21</b>
5.1 Technologies . . . . .	21

5.1.1	Frontend . . . . .	21
5.1.1.1	SPA . . . . .	21
5.1.1.2	Angular . . . . .	22
5.1.1.3	Node Package Manager . . . . .	22
5.1.1.4	ChartJS . . . . .	22
5.1.2	Backend . . . . .	23
5.1.2.1	Java . . . . .	23
5.1.2.2	Database . . . . .	23
5.1.2.3	Serialization . . . . .	23
5.1.2.4	Swagger . . . . .	24
5.2	Coda API . . . . .	24
<b>6</b>	<b>Evaluierung</b>	<b>26</b>
<b>7</b>	<b>Conclusion and future development</b>	<b>27</b>
	<b>Bibliography</b>	<b>29</b>
	<b>List of Figures</b>	<b>30</b>
	<b>List of Tables</b>	<b>31</b>
	<b>Abkürzungsverzeichnis</b>	<b>32</b>
	<b>Anhang</b>	<b>33</b>
A	Anhang: Quelltexte . . . . .	33

# Chapter 1

## Introduction

Artificial intelligence (AI) has in recent years become one of those hype words, evoking everything from sceneries of certain doom at the hand of uncontrollable robots to visions of plenty, where ai puts us on a path to enlightenment. Much more quietly if perhaps not quite as grand AI has indeed changed how we live our lives, recommending songs to hear and items to buy, steering cars and investing money, making homes listen to their inhabitants and beating world champions in their chosen field.

As a branch of AI, machine learning (ML) has, aided by the ongoing digitalization and the generation of big data, become a strong tool in the toolkit of modern data analyzation and software development. It allows a completely new level of automation, having a program not only follow a set piece of rules to their conclusion but to create those rules in the first place.

With supervised learning, programs may now find underlying rules behind large data collections and use it to label new input. Is an email spam, is a patient ill and how much is your car worth? Questions that, given the right information and training a program can now predict with high accuracy.

That is the point where automation today comes to its end. Finding a fitting ML algorithm and configuring it to resolve a problem is a task that in most cases must be done by ML experts. And just as diverse as the problems are that may be solved with ML and supervised learning are the ways to configure these algorithms, making this a nontrivial challenge.

To automate this process, GT-ARC (German Turkish Advanced Research Center for ICT) has started CODA, a fundamental research project in algorithm selection and hyperparameter optimization. [1]

CODA, however, does not come with an easy to use visual user interface (UI). With Sparked an interface to the CODA project will be created, allowing ML enthusiasts and

specialists to use the developed solutions and giving the team an interface to demonstrate CODAs capabilities.

# Chapter 2

## Basis

In this section, the basic concepts and terms needed for further discussion will be introduced. This includes general ML terminology and how it applies to Sparked, as well as terminology of Sparked that based on these fundamentals. It is important to note, that Sparked itself does not contain any ML functionality. All ML capabilities come from CODA. Because of that, CODAs terminology will be adopted.

Supervised learning tries to approximate a function from a data point to a value by learning the underlying rules from data, where the corresponding value is already known. There are two types of supervised learning problems, discrete ones, like labeling mail into spam or no spam and continuous problems, like predicting the price for a used car.

Problems that fall in the first category are called classification problems, whereas problems with continuous values are called regression problems. While CODA today supports both classification and regression problems, it comes from a classification background. Therefore the naming that Sparked and this work use is that of classification, even though both types are supported.

To learn a supervised learning algorithm needs a dataset that has already been labeled. Within the context of Sparked, a dataset is the data from which the program is supposed to learn the underlying rules.

The underlying algorithm to evaluate the dataset with is the classifier. Since Sparked does not differentiate between classification and regression problems, such an algorithm may be a classification or regression algorithm even though the name might suggest the former. Classifiers may have additional information attached, hyperparameters the classifier needs to further adjust it to the given problem. These hyperparameters have a special place in ML because they are typically a fixed value set at the start of evaluation and not learned or adjusted from the ML program. Optimizing these hyperparameters



is a major concern in ML and automating this optimization one of the main goals of Auto-ML projects like CODA.

Part of the training is validating the output. In its essence, that means taking a part of the dataset using it not for training but have the algorithm predict the known labels after training and checking how good the predictions were. This is helpful not only to put a number on how good the training was but also helps to discover cases of overfitting, which is always a concern with supervised learning. Since training data should not be used for validation, every point of validation data reduces the number of data points the program has to train. Getting the right balance between validation and training is what the validation method is for.

The first step to evaluate how good the ML training was, is to define what good is in the context of the current problem. For example with credit card fraud data, where almost all transactions are non-fraudulent it is not enough to measure the accuracy or how many predictions were correct, as just predicting all of them as good would be incredibly accurate, but not give any valuable information. The measure to apply must be supplied by the user and is called the metric or target metric within Sparked.

Combining these four values, dataset, classifier with hyperparameters, validation method and metric is the information Sparked needs to start an evaluation with CODA. Such an evaluable set will be called Task.

The focus of this Sparked is however not to solve ML problems or Tasks, but to compare multiple Tasks against each other, that only differ in the classifier, its algorithm or hyperparameters. Such a set of Tasks will be called an Order. The Order is the main object in Sparked. All actions will be done on an Order.

## 2.1 State of the art

There are several interesting projects when it comes to visual interfaces for ML software. One of these is the web application OpenML. Standing for open machine learning, OpenML [www.openml.org] gives access to ML tools to a broader audience including readily available datasets and predefined classification algorithms. Looking at this project can show how some of the hurdles in developing Sparked may be solved.

The general look and feel of OpenML is very bright with strong vivid colors and hard corners. Separate values are often put right next to each other, only separated by a dash. This gives the UI a fresh and tidy look and allows a lot of information to be put on a small amount of space, but it also makes it hard to identify entries at a glance.

## Create new task

### Choose Task Type

**Task types**

Supervised Classification

In supervised classification, you are given an input dataset in which instances are labeled with a certain class. The goal is to build a model that predicts the class for future unlabeled instances. The model is evaluated using a train-test procedure, e.g. cross-validation.

To make results by different users comparable, you are given the exact train-test folds to be used, and you need to return at least the predictions generated by your model for each of the test instances. OpenML will use these predictions to calculate a range of evaluation measures on the server.

You can also upload your own evaluation measures, provided that the code for doing so is available from the implementation used. For extremely large datasets, it may be infeasible to upload all predictions. In those cases, you need to compute and provide the evaluations yourself.

Optionally, you can upload the model trained on all the input data. There is no restriction on the file format, but please use a well-known format or PMML.

### Task inputs

**Dataset(s)**

**Target Feature**

**Estimation Procedure**

10-FOLD CROSSVALIDATION

**Specify cost matrix (optional):**

**Evaluation Measures**

**SUBMIT**

Figure 2.1: OpenML UI, clean and colorful with a lot of tight packed information. The basic features of OpenML allow the creation of tasks.

The lists are not paged and can show all results at the same time, using on the fly loading of additional items while scrolling. While this should make it very hard to find any data, the usage of filters and search bars helps find a value fast. Very notable is the usage of likes, reach and impact values, giving the page a social media vibe. This is part of the core idea of OpenML, to facilitate sharing and reuse of ML tools, but has no correspondent feature in Sparked.

While creating a task, the user will choose what kind of machine learning they want, like supervised learning, supervised regression and clustering, an estimation procedure which maps to the validation method in Coda and the evaluation measures. All these input steps are situated on the same view. As selected values stay visible, a overview over the selected data is available at all times.

Like Sparked, AutoML provides a view to see the evaluation measures. Here the concrete values are displayed as well as a graphical representation. AutoML does not differentiate between the measures and shows all of them on one site with help of the JavaScript chart library highcharts. This sometimes led to performance issues in the browser, making the page appear sluggish, on some occasion even freeze. It should

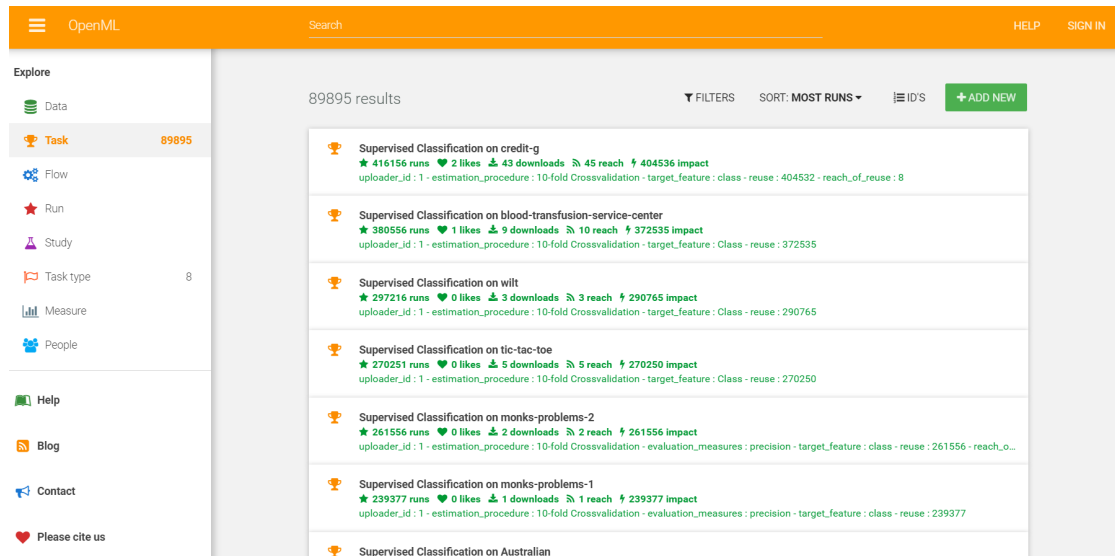


Figure 2.2: Task Creation in AutoML requests most of the data an order creation in Sparked will need.

be kept in mind, that even good JavaScript chart libraries can have performance issues, especially while loading and scaling multiple charts at the same time.

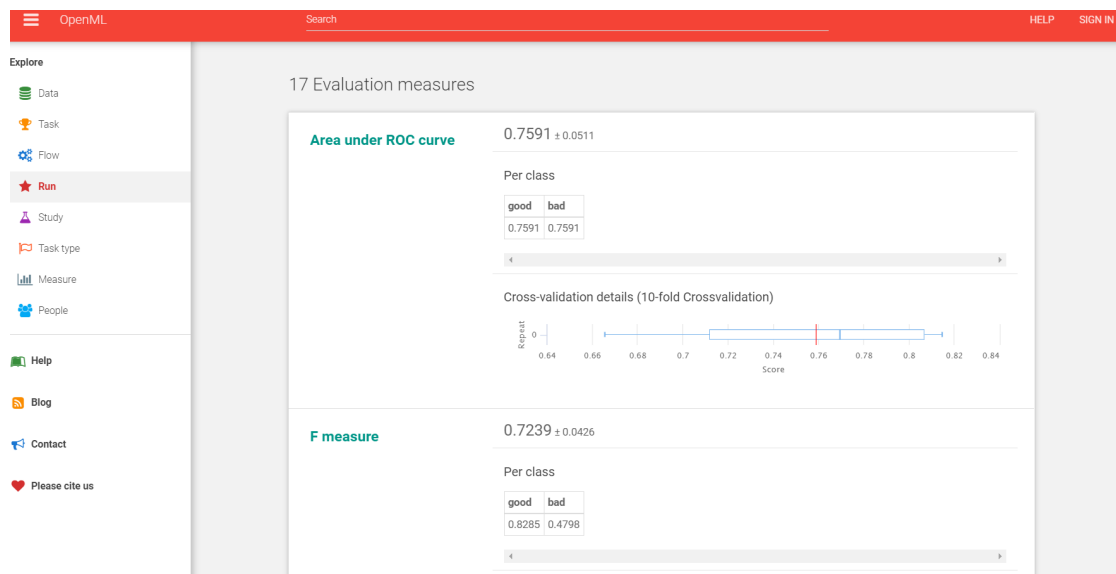


Figure 2.3: AutoML displays several evaluation measures at the same time with graphical components using Highsofts charting JavaScript library highcharts.

# Chapter 3

## Problem analysis

### 3.1 Requirements

One of the surprisingly hard tasks in software development is the requirement analysis. In this section the reason for development will be described and a list of functional and non-functional requirements given, to further define the scope of this project. At the end a target audience will be proposed for which this software is being written.

**At the start of this project a couple of top-level requirements had been given.**

Create a UI for the existing software CODA to showcase CODAs capabilities to give visual support in talks and demonstrations. The main use case for Sparked is clear. It is to take the capabilities of CODA in the form of its web api and create an interface on top. An important factor here is the target audience, people who are very tech savvy and familiar with machine learning. Equally important is the mode of consumption, not open for general use, but either by a single user as the lecturer or a small group of users for a limited amount of time.

**Sparked should support a simple and configurable startup.**

Running sparked locally or connecting it to a different instance of Coda should be possible changes in code and without recompiling.

**Allow for a clean slate startup.**

Since its use case is to run for presentations, it is required to start an instance without any old data in it, a clear state, that will always look and function the same. Such a start should create all necessary files, database tables and related items, removing any remnants of earlier runs.

**Sparked is a web application that runs in docker on a Linux system.**

For Sparked it had been decided, that it may run as a web application to be accessed via browser and that the server side should be hostable in docker on Linux to ensure compatibility.

**Not for productive use.**

Reducing the scope of the project in key areas such as security and user management.

**3.1.1 Nonfunctional Requirements**

It is important to note, that CODA is not a finished product, but a project that is currently in development. That includes both soft changes like adding new datasets or classifiers to the existing ones, but also changes to endpoints or the structure of data. From this the first non-functional requirement arises:

- Create Sparked in a way, that favors extensibility.

Using Sparked as a demonstration object in presentations necessitates a certain UI paradigm:

- UI must be visible displaying via beamer in a not ideally lit room.

Directly from the main objectives it can be concluded that

- All configuration should be doable via config files without recompiling the project and
- Necessary files and database tables must be automatically created as needed.

**3.1.2 Functional Requirements**

Functionally Sparked needs to support the capabilities that the CODA api publishes.

- Creating an order
- Starting an order
- Listing existing orders
- Displaying the evaluation data of a completed order

## 3.2 User Personas

When trying to create a user interface, it is necessary to think of the users that will interact with the system and try to see the software through their eyes. A helpful tool in that regard is the use of user personas, one or more users that represent the archetypes of all persons interacting with the system. Ideally this would begin by gathering information on users, evaluating web analytics data, interviewing real users and conducting surveys. Unfortunately, this goes beyond the scope of this project which leaves the possibility to create personas from assumptions about the userbase alone. This has the danger to reinforce already existing biases and does nothing to validate the existing assumptions on scope and usage of the project but is still worthwhile, helping to view the design process through the lens of different people. The goal in creating personas is to represent much of the user base and have a persona for every major user group. For Sparked these groups are:

- The presentation viewer
- The presenter
- The developer

### **Nora, 26**

- Graduate student
- Advanced knowledge in machine learning
- Wants to get an understanding of CODA
- Watches a presentation about CODA

Nora had an interesting discussion about automated hyperparameter optimization with a CODA team member and has been invited to view a presentation to learn more about the project. She is knowledgeable in AI and works on a research project in machine learning, but not specifically in hyperparameter optimization.

### **Jack, 27**

- Researcher
- Self-taught knowledge of server systems

- Advanced knowledge of machine learning
- CODA expert
- Wants to top of his CODA presentation with a hands-on example

Jack works with CODA on daily basis. He knows all ins and outs of the program and has advanced knowledge in ML. He has some experience in server systems but more from necessity than from interest. He likes efficiency and generally only has a small amount of time to prepare for a presentation. In preparation for his presentation he needs to setup a new instance of Sparked, since the last one has been shut down last week when it was not used.

#### **Tim, 23**

- Writing his bachelor thesis
- Only minor knowledge in machine learning
- Strong foundation in programming languages learned in university
- Some experience programming outside of university in a handful of small personal projects

Tim has been tasked to change Sparked to support the changes that have happened in Coda over the course of 2 years. He likes to code and has already created a basic android app and several web apps in his free time. He has little experience in expanding on a foreign codebase. These Personas represent the target audience and will be used to view Sparked through the eyes of the user.

### **3.3 Summary**

This section has defined both the top-level requirements and the target audience while proposing a way to take the information on the target audience to help in future decision-making using personas. Top-level requirements can be listed as:

- Create and start an order
- List existing orders
- Display the evaluation data of a completed order



- Facilitate extensibility of the codebase
- UI must be visible displaying via beamer in a not ideally lit room
- All configuration should be doable via config files without recompiling the project
- Necessary files and database tables will be automatically created as needed
- Frontend is a website
- Server side is running in a Linux docker container

The target audience has been defined and representational user personas been created:

- The presentation viewer, represented through Nora
- The presenter, represented through Jack
- The developer, represented through Tim

# Chapter 4

## Concept

In the last section the requirements for this project have been discussed, creating an understanding of the functional as well as nonfunctional requirements. Additionally, a way to understand the target audience using personas was prepared.

In this section this information will be taken to create the visual and architectural foundations of Sparked. At the end a roadmap should exist for the development process in the form of a design guide with guidelines on how to create the UI and a workflow analysis, creating an understanding which steps will be needed to meet the functional requirements as well as a shallow architectural design for the workflows specifically and the whole application in general.

### 4.1 Visual guidelines

Before diving into the architecture, some general guidelines for the UI should be created. These are not workflow specific but instead are mostly dependent on the target audience, which is represented by the three personas created previously:

- Nora is part of the audience, viewing a Jacks presentation
- Jack holds a presentation to demonstrate CODAs capabilities
- Tim is a developer tasked to expand Sparked

#### 4.1.1 Looking from the users' point of view

**Focus the attention.**

Need: Jack wants to create an order in a talk. He can not give his whole focus on

the process, as he is mainly invested in relaying information to the audience. He does not want the UI to steal the focus of the audience from him, allowing him to keep Sparked open even while speaking. Conclusion: Reduce the amount of data displayed at a given time. The user will necessarily be focused on a specific item if it is the only one on the screen. Hiding or graying out other elements will be used if possible. Similarly having long lists with many items holds the focus of the viewer for a long time. Reducing the number of items visible at any given time, using bigger fonts will reduce the time the viewer needs to take their focus back to the presentation, even if this means that the information density is reduced, and more scrolling is needed.

### **Good readability.**

Need: As a viewer of the presentation Nora wants to be able to follow what is visible on the screen, even if lighting is not favorable or she is sitting on the far end. Conclusion: A high contrast color scheme should help readability. Using bigger fonts and lines with higher thickness will help as well.

With these requirements in mind a set of design goals can be formulated.

- **Reduce the information density**

In many UIs the goal is, to put as much information and functionality into as little space as possible. A good example of this trend is the UI of OpenML, where the font becomes small and data points are only separated by a dash, not by a visual element. On a computer screen this will allow a user to see most relevant data without switching in and out of detail pages, reducing the amounts of mouse interactions they must do. On a beamer could easily become a distraction, taking the focus of the audience away from the presentation while they are trying to decipher the small fonts.

- **High contrast, big font**

This too is a way to make the UI as readable as possible. Especially if the UI is displayed on a beamer, this becomes important, as lighting conditions are often not optimal.

- **Use a strong color to guide the attention**

Color is a good way to grab the attention of the viewer. Having a selected item light up with a strong color can be used like a flashlight to indicate the important pieces of information. Trouble arises, when too much color is used. If several pieces are strongly colored, it loses its potency to grab the attention.

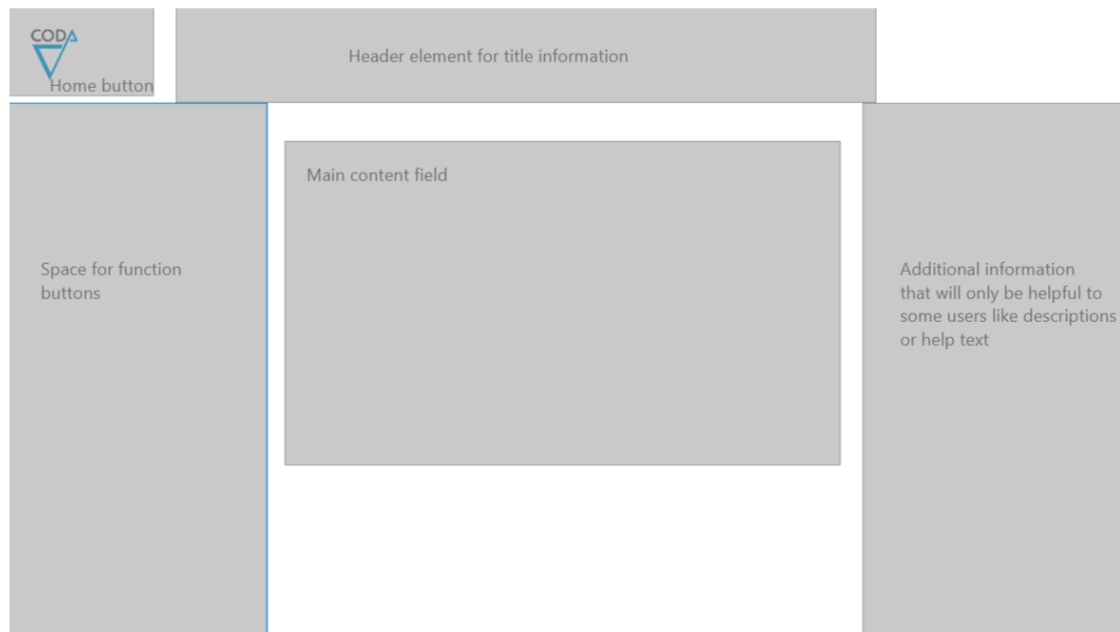


Figure 4.1: Segmenting the UI

- **Use the same look and feel throughout the application**

Using the same places for related purposes on all views and reusing visual elements in the entire application and using uniform font and color scheme, allows the user to become familiar with the UI faster, which is in most cases wanted by the designer.

The first step in creating a common look and feel is to create a common map of where what kind of information goes on the screen. The first concern is to keep the UI close to what the user would expect, which means putting the icon in the top left corner and title information close to the top.

For this design the center will be used for main information, like the list of orders or the diagrams for the evaluation view. On the left side, functions like ok and cancel will be situated. Whereas information text, that may describe certain functions or data will be on the right.

## 4.2 Architecture

Armed with an understanding of what Sparked is set out to accomplish, a rough architecture will be created. Software architecture is a very wide field that may be used to define a program down to a level only just above the implementation itself. For this project a rough overview of the components however should be enough. Creating detailed UML (Unified Modeling Language) descriptions of an application is very time consuming and

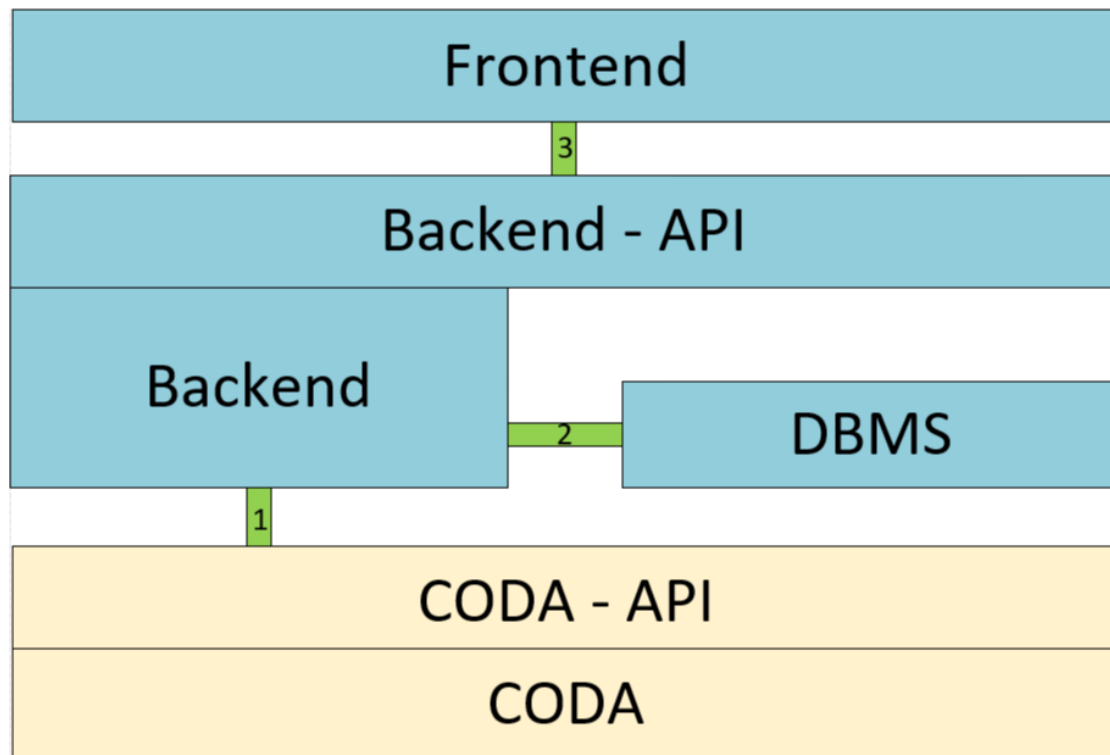


Figure 4.2: 30000-foot view of Sparked architecture

it always needs to be calculated whether the development speed or code quality gained is worth the amount of work put into documenting an architecture beforehand.

### 4.2.1 The 30000-foot view

To start with it is helpful to view the whole project from a distance. Creating a first high level architectural description, often called the 30000-foot view, will help assess the core concepts and major building blocks.

In this diagram yellow parts are provided. White space indicates the possibility to distribute parts on different systems. Green lines show points of interaction between the different parts. A point of interest is the CODA API and the interaction line 1, since this is the part that is defined outside of Sparked. This means, that the technology and structure are already known. The API itself is a REST API, using JSON (JavaScript Object Notation) for data transfer. Its endpoints deliver the building blocks of all further development. There are four endpoints delivering the supported classifiers, validation methods, datasets and evaluation metrics. An endpoint with the status of all orders, one to get the status of a specific order and one to get the results for a specific order and an endpoint each to start a task or an order. [For a more comprehensive description see the



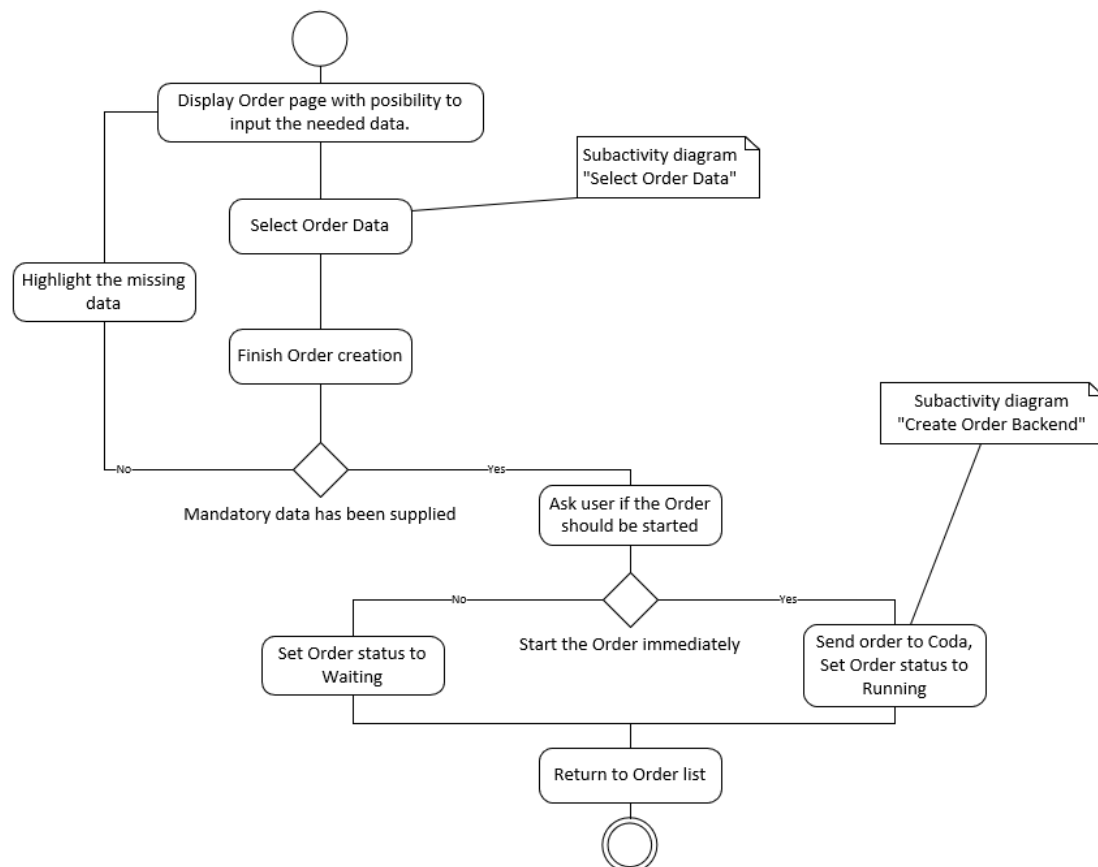


Figure 4.4: Activity Diagram – Create Order

- The data input or selection
- Create the Order and validate whether necessary data has been supplied
- Check if Order should be started immediately
- Start the order

One of the mayor scenarios is to show example orders in a presentation or similar format. Creating and starting an order from scratch in such a situation has the problem, that the runtime for machine learning tasks can be quite substantial. This would mean finding the correct order out of a possible longer list in a short amount of time. To support this, it would be helpful for the user to give a kind of identity to an order. Since Sparked has its own data storage, it is possible to add additional data to an order, that is not supported by CODA itself. For this scenario it was decided, to have an additional optional title field. This does not change the fundamental workflow, only adds an additional step in selecting the order data, but also shows a way for future expansion. If ever concepts like labeling of orders or user accounts may become relevant, they can be achieved with the same concept as the title field.

#### 4.2.2.2 Workflow: Listing orders

Once orders can be created, showing and opening all existing orders becomes the next workflow to consider. Its main goal of this workflow is:

- See all orders created by this instance of Sparked and to
- Open detail page of a single order
- Open evaluation page of completed evaluations and display the evaluation data returned by CODA

The first part is rather straight forward from a UI point of view. Get a list of orders and display some of its data in a table like view. There are two points to consider: The first is, that the status information of an order is an important information in this list. The status is however subject to change and its current value can only be supplied by the CODA backend. This means a UI request to display the order list must receive data not only from Sparked but from CODA as well. This has the potential to be slow and must be considered when implementing. A way around would be by loading the data beforehand in an asynchronous manner.



Ideally this would be done by having the CODA API push the data or a data changed information towards Sparked whenever a status has changed. The CODA API does not allow for this though, making a polling approach the only viable option for preloading. Since the data will contain information on all orders this may in future become a large and costly request. At the same time the information should not become too old as for the user to wait on an order that has already been finished. As a good polling timer is rather subjective this is a time a value should be changeable via the application properties.

The second problem is on which orders to show. Both the CODA API as well as Sparks own database may contain orders, but an order that was never started will only appear on the internal database, while an order that was started with for example a different Sparked instance, will only be available via the CODA API.

One of the starting requirements is, that Sparked must support a clean slate startup. Since deleting orders via the CODA API is not possible and probably not wanted, the decision was made, to only show orders that have a corresponding entry in the applications own database.

#### **4.2.2.3 Workflow: Evaluation**

The last workflow is the evaluation. Here an order is open XXXXXXXXXXXXXXXXXXXX

# Chapter 5

## Implementation

### 5.1 Technologies

With a clear picture on the requirements and the aspired architecture, it is time to talk about technology. What Programming languages will be used and what are the main libraries.

#### 5.1.1 Frontend

One of the requirements is for Sparked to be a web application. Therefore the frontend will ultimately be in html, javascript and css, the web stack. With Flash and Silverlight support finally ending in 2020 and 2021 respectively [2] [3] there really is no alternative. There are however several ways how to get to this html, javascript and css.

##### 5.1.1.1 SPA

There are numerous ways to create dynamic websites. One of them is the use of a Single Page Application (SPA) framework. SPAs work in such a way that the main page is only loaded and the Document Object Model (DOM) only evaluated once by the browser. Afterwards javascript is used to make server calls, animate object, bind event handler to actions, change the DOM and do everything else that is required for the website to function. This means that the UI is effectively been built in the browser on the client machine.

### 5.1.1.2 Angular

For this project the choice of SPA framework fell on Angular. This is not for any specific technological reason, as all the well-known SPA frameworks can support anything needed by this project, instead the choice was made because of previous knowhow existed.

Angular is maintained by google, was first released in 2009 [7] and is available under a modified MIT license [8], with the newest Version being 8.0.0, released on May 28th, 2019 [9]. Sparked uses Version 7.2.3 as it was the newest at development time.

Angular uses TypeScript as programming language. TypeScript is a superset of JavaScript extending it with features like inheritance and strongly typed variables. It being a superset means that all valid JavaScript code is also valid in TypeScript. That is not true the other way. Instead TypeScript uses a preprocessor to compile TypeScript code into JavaScript before deploying it to the client.

TypeScript is an open source project available under the Apache License 2.0 [10] and was first released by Microsoft in 2012 and exists now in Version 3.4 [11]. Sparked uses the Version 3.2.4.

### 5.1.1.3 Node Package Manager

When working with angular it is recommended to use the NodeJS and the node package manager (npm) to install and manage angular and the libraries used by angular.

### 5.1.1.4 ChartJS

As seen in the previous chapter, Sparked needs a way to display data as graphs. There are a lot of charting frameworks available. From commercial libraries such as Highcharts, over free software like google charts to free open source projects like Chartist. There are some characteristics for a charting framework to be considered in this project. The library should be free to use without constraints like watermarks. A couple of chart types must be included and while scatter-, line- or box charts are rather standard, box and whiskers plots are only supported in some libraries.

For this project the decision fell on ChartJS. It is available under the MIT license [12], can be used with the node package manager and has an active community behind it [13]. It supports all standard plot types and with the help of an extension, chartjs-chart-box-and-violin-plot, it supports box and whiskers plots. The version used is 2.8.0.

## 5.1.2 Backend

Having looked at the client-side technologies it is now time to do the same for the server side.

### 5.1.2.1 Java

Almost every major programming language today supports natively or has libraries, that allow to create a simple CRUD webserver. As such the question which language to use is not so much dependent on the features of the language itself but more on the level of knowledge current and future developers might have in it. With this Java becomes a save bet as it is one of the most widely used programming languages [15].

The Java backend needs to supply data and functionality for the frontend SPA. Using Spring Boot Using Spring Boot in version 2.1.1 allows to easily create a Rest interface

### 5.1.2.2 Database

For data storage MongoDB, a document based no-sql database, was chosen. MongoDB is a tried technology with existing drivers for Java. The main benefit of a schema-less database like MongoDB is, that changes in the data do not need to be consciously reflected into the database. This makes changes to Sparked easier, as it reduces the points in code that need to be changed if the CODA API where to change. For the database driver, mongodb-driver in version 3.8.2 was chosen.

### 5.1.2.3 Serialization

The Sparked backend gets data from the Coda API on one side and sends data to the UI on the other. Both are sent in JavaScript Object Notation or JSON format. To convert said data from JSON to what is often called a POJO, a plain old java object, to work with it on the Server and back to JSON, a serialization function is needed. Using Jackson this process can be automated converting JSON objects to Java objects without the need to write converters. Jackson will instead try to convert these automatically using reflection and a combination of conventions and attributes to map JSON to Java values. This has the added benefit, that a change in the underlying API in many cases only needs for the transformed object to be adapted. In comparison the change of a handwritten serializer or deserializer would take up much more time.

### 5.1.2.4 Swagger

The Sparked backend publishes an API for consumption by the angular frontend. While this is an internal API and does not technically need to be documented, it will be helpful for future developers to see all valid endpoints and their usage in one spot. Swagger is the tool of choice when it comes to API documentation. It creates a page at the relative path *swagger-ui.html#/* that shows an overview of all endpoints grouped by controller, supported parameters, responses that can be expected and the data structures of return values. This information should enable a frontend developer to create a change a UI or even create a new one without having to go into the code themselves.

## 5.2 Coda API

The machine learning functionality of Sparked comes fully from CODA via the CODA API [See Appendix A]. The CODA API returns JSON Objects, which is where Jackson comes into play. Jackson maps objects from JSON to the corresponding Java object. It makes the connection either by property name or by using attributes.

```

1  {
2    "id": "class specific specificity",
3    "highValueBetter": true,
4    "isScalarMetric": false,
5    "isClassSpecific": true
6  },

1  @JsonIgnoreProperties(ignoreUnknown = true)
2  @JsonInclude(JsonInclude.Include.NON_NULL)
3  public class Metric {
4      private String id;
5      private Boolean highValueBetter;
6      private Boolean isScalarMetric;
7      private Boolean isClassSpecific;

```

This is a fast and reliable way to serialize and deserialize the data. Since it maps by using known names, this method only works with known named properties. For properties where the property name is not constant, this does not work. CODA for example puts an id before the classifier parameters. For this Jackson supports custom deserializers. Code example: The

```

1  "results": {
2    "bestConfiguration": {
3      "params": {
4        "dtc_6b195ee4e3bb__seed": "159147643",

```

```

5         "dtc_6b195ee4e3bb__cacheNodeIds": "false",
6         "dtc_6b195ee4e3bb__checkpointInterval": "10",
7         "dtc_6b195ee4e3bb__impurity": "gini",

```

Having a component do what otherwise would be many hundred lines of custom code is already worth adding a library, but the main benefit here becomes apparent, when trying to expand the CODA data structures. Because the database is schema less and the frontend is compiled into JavaScript, neither of these parts need to be changed to support additional fields in CODAs data structures. The only change needed is to create matching variables as well as getter and setter functions in the corresponding java class and the values should become accessible in the frontend. Of course, this only works on auto mapped classes. For the evaluation result data both the matching class as well as the deserializer, `EvaluationResultMetricDeserializer`, must be edited. Changing the deserializer is not the biggest issue, but it is to be noted, that for an earlier version of CODA API several deserializer classes existed with one having more than a hundred line of low level code to traverse and read json nodes. Changing a class like that can quickly become hard and is an easy place to make errors. Configuration XXXXXXXX CHANGES HERE To load the properties, the standard java class `Properties` (`java.util.Properties`) is used.

- Spring Boot Startup - Logging - Properties - MongoDB - Dependency Injection - Interface Usage? - Controller / API Description
- General Structure o Order Generator o Orders o Order o Landing Page o Order Generation Service o Navigator Service o Backend Service - DTOs - ChartJS - Sass? - Translation - Listing Package.json

# Chapter 6

## Evaluierung

Länge: ca 1-5 Seiten

Sind die gesteckten Ziele zur Problemlösung durch die Implementierung erreicht worden? Was kann die vorgestellte Lösung, was kann sie nicht. Des Weiteren gehören zu einer Implementierung auch immer Tests, ob die Implementierung erfolgreich war! Diese Tests müssen auch dokumentiert werden. In diesem Kapitel sollte daher eine Beschreibung des Aufbaus und der Ergebnisse von Testläufen/Simulationen vorhanden sein. Ebenso sollte eine Interpretation der Ergebnisse die Tests abschließen. Es ist auch wichtig, nicht nur positive, sondern auch negative Ergebnisse zu dokumentieren und zu interpretieren.

## Chapter 7

### Conclusion and future development

Creating Sparked became an interesting experiment in the difference between knowing a technology versus having mastered it. Being able to check of a list of technologies and coding paradigms that should go into a web application undoubtedly helps to know what needs to be developed, but having the actual knowledge how specific libraries work is a different thing altogether and getting stuck on simple things like dependency injection with an unknown framework or learning the ins and outs of how a specific logger has to be configured is both tedious and very time consuming.

Looking at the program, the Sparked backend does what it is supposed to do. It is created with modern but tried technologies, the code is reasonably clean and should be simple enough to extend. On the front side of the program this becomes more complicated. Angular offers a lot of tools to structure its code for reuse, which have only been used to a small degree. Here the code quality would have benefited from a real frontend developer with Angular experience.

That said, the frontend works, it connects to the backend to collect the necessary data offers and offers the functionality the requirements asked. The UI has been designed in a way to simple, easy to use and good to read.

But while functionality can easily be tested by creating and starting an order and once finished looking at its evaluation page, assessing the programs look and feel can not be done by a single person, as it is rather subjective. Because of time constraints a qualitative evaluation of the user interface was not possible, the readers must make their own assessment.



In the future the UI would probably benefit a lot from a domain expert going over the different views, deciding what information the user will need on each of them. It is often the simple things, like what fields are needed to quickly decide if a user would want to open an order out of the 20 other visible orders, that give a big benefit to the usability. All data should be there right under the hood and having that domain knowledge should allow for an easy change.

Next to this, simple search capabilities on all lists would make a marked improvement. Even now there are lists that are long enough to warrant searching, with continued development of CODA and usage of Sparked this will become more pressing as list become longer.

All in all while, not a full success, the project can be considered a good stepping stone towards to create a UI that can adequately represent such a powerful tool as CODA.

# Bibliography

- [1] GT-ARC: *GT-Arc Homepage*. <http://www.gt-arc.com/?lang=de>. 1
- [2] MICROSOFT: *Microsoft product lifecycle*. <https://support.microsoft.com/en-us/lifecycle/search?alpha=Silverlight%205>. 21
- [3] COMMUNICATIONS, ADOBE CORPORATE: *Flash & The Future of Interactive Content*. <https://theblog.adobe.com/adobe-flash-update/>. 21

# List of Figures

2.1	OpenML UI, clean and colorful with a lot of tight packed information. The basic features of OpenML allow the creation of tasks. . . . .	5
2.2	Task Creation in AutoML requests most of the data an order creation in Sparked will need. . . . .	6
2.3	AutoML displays several evaluation measures at the same time with graphical components using Highsofts charting JavaScript library high- charts. . . . .	7
4.1	Segmenting the UI . . . . .	15
4.2	30000-foot view of Sparked architecture . . . . .	16
4.3	Data needed to define an Order . . . . .	17
4.4	Activity Diagram – Create Order . . . . .	18

## List of Tables

# Abkürzungsverzeichnis

<b>HTML</b>	Hypertext Markup Language (Textbasierte Webbeschreibungssprache)
<b>CSS</b>	Cascading Style Sheet
<b>UML</b>	Unified Modeling Language
<b>REST</b>	Representational State Transfer
<b>SPA</b>	Single Page Application
<b>UI</b>	User Interface, often used interchangeable with GUI
<b>API</b>	Application Programming Interface
<b>AI</b>	Artificial Intelligence
<b>ML</b>	Machine Learning
<b>Auto-ML</b>	Automated Machine Learning
<b>GT-ARC</b>	German Turkish Advanced Research Center for ICT
<b>ICT</b>	Information and Communication Technology
<b>JSON</b>	JavaScript Object Notification
<b>DB</b>	Database
<b>DBMS</b>	Database Management System
<b>GUI</b>	Graphical User Interface
<b>IP</b>	Internet Protocol
<b>NPM</b>	Node Package Manager

# Anhang

Hier befinden sich für Interessierte Quelltexte sowie weitere zusätzliche Materialien wie Screenshots oder auch weiterführende Informationen.

## A Anhang: Quelltexte

### Beispiel.java

```
1  class Beispiel{  
2  
3      public static void main(String args[]){  
4  
5          System.out.println("Hello_World");  
6  
7      }  
8  
9  }
```