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

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.

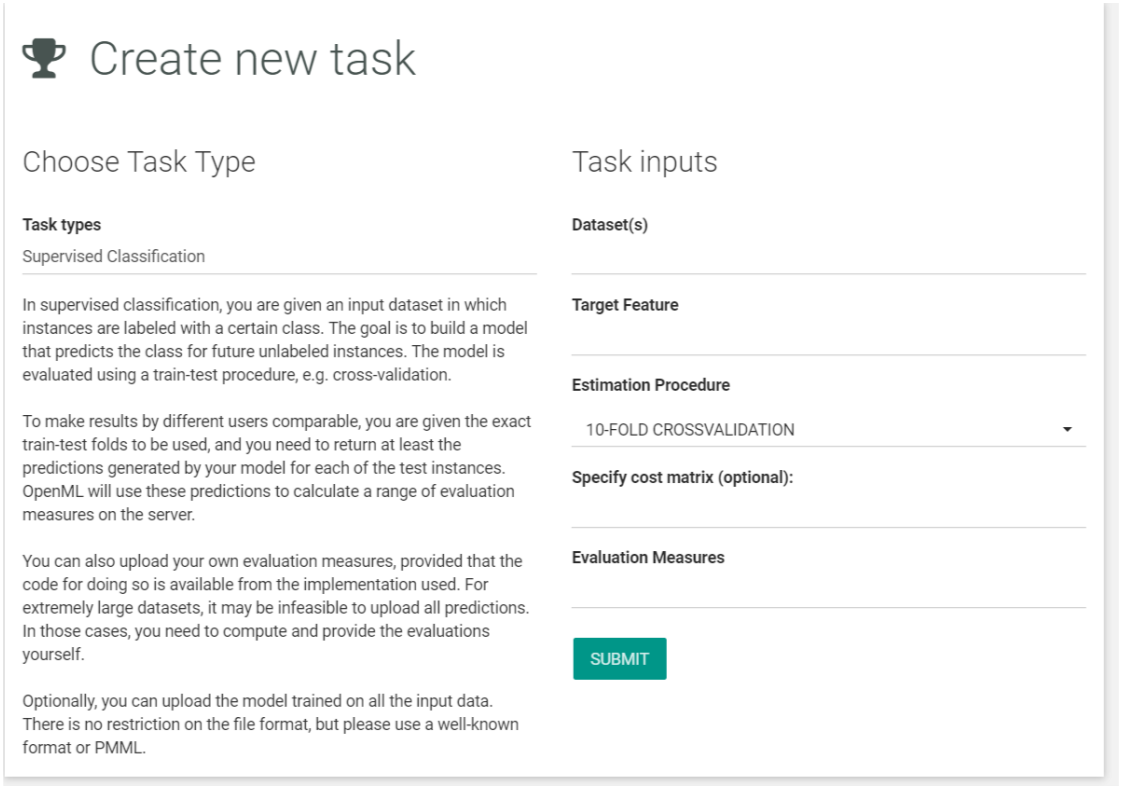

Figure 2.1: OpenML UI, clean and colorful with a lot of tight packed information. The basic features of OpenML allow the creation of tasks.

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

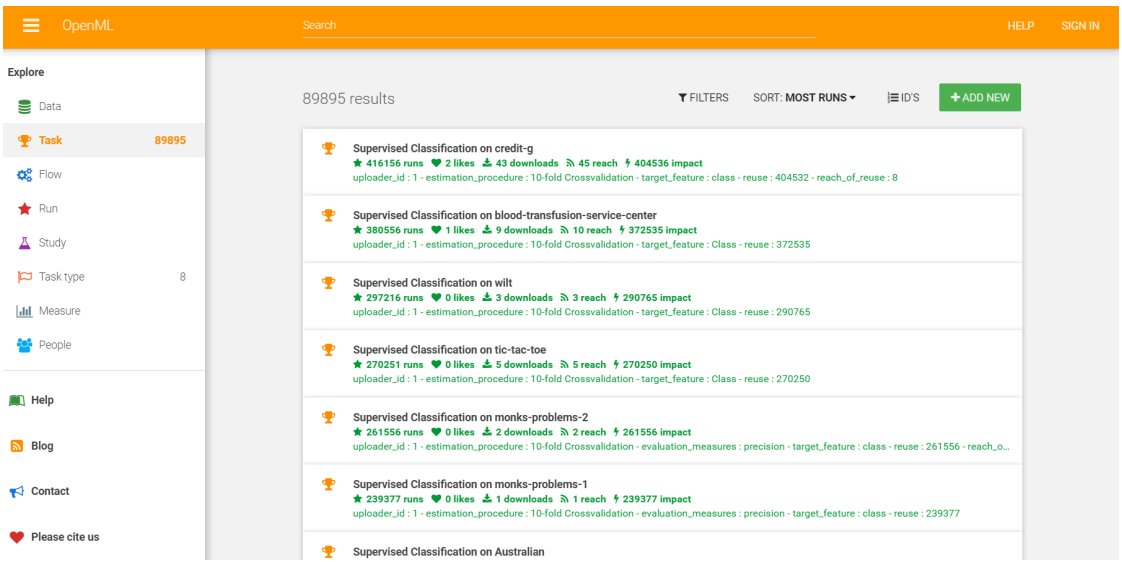

Figure 2.2: Task Creation in AutoML requests most of the data an order creation in Sparked will need.

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.

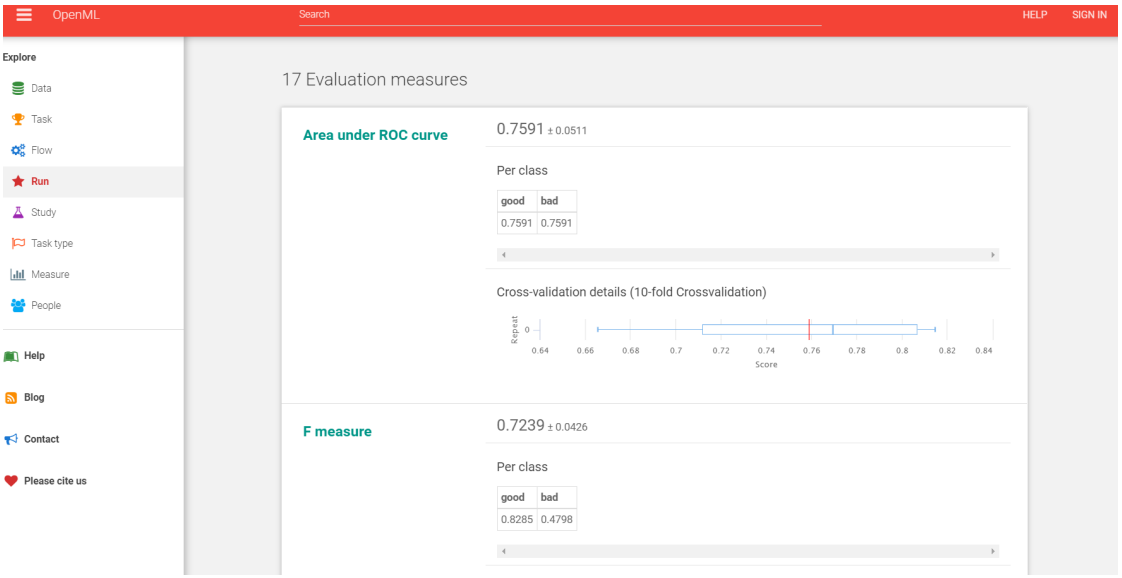

Figure 2.3: AutoML displays several evaluation measures at the same time with graphical components using Highsofts charting JavaScript library highcharts.

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. In 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.

Showcase CODAs capabilities to give visual support in talks and demonstrations. The main requirement for Sparked 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 ML. 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.

Sparked should support a simple and configurable startup with the possibility for a clean slate startup.

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:

- The 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 will be automatically created as needed.

Functional Requirements Functionally Sparked needs to support the capabilities that the CODA API publishes.

- Create an Order
- Start an Order
- List existing Orders
- Displaying the evaluation data of a completed Order

3.2 User Personas

When trying to create a UI, it is necessary to think of the users that will interact with the system and try to see the software through their eyes. The use of user personas, one or more users that represent the archetypes of all persons interacting with the system, helps to think from a user perspective and has been shown to support the creation of user-friendly interfaces [2].

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 of creating personas is to represent most of the user base and have a persona for every major group of users. 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 ML, 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 daily. 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 set up 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 two 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
- The 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
- The 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 cannot 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, lists with many items hold 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 the 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 the 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 as a flashlight to indicate the important pieces of

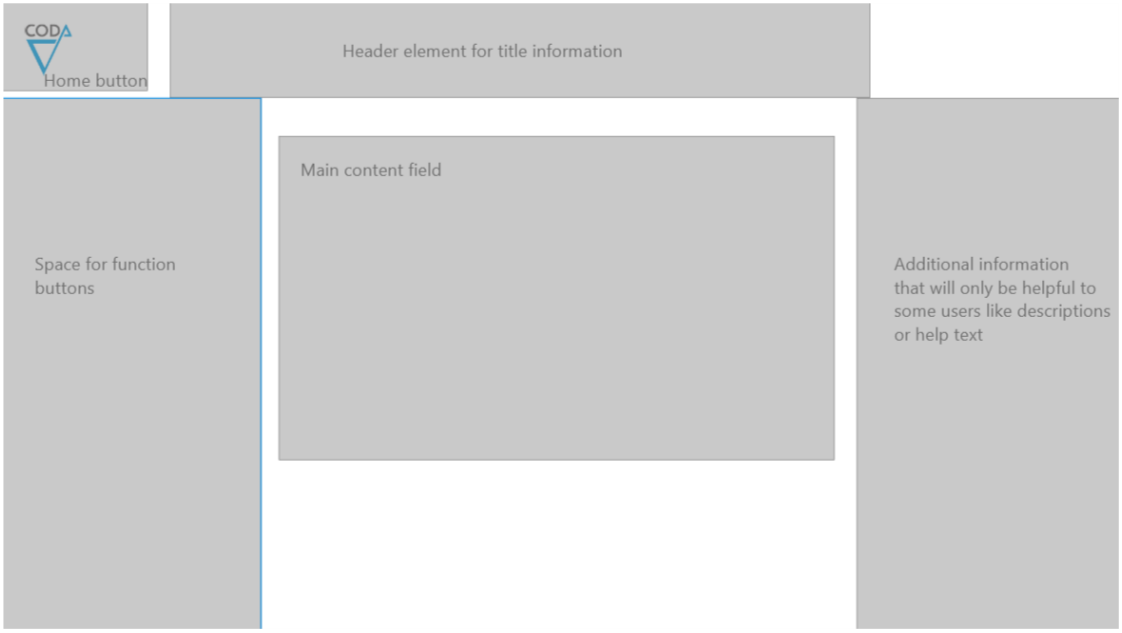

Figure 4.1: Segmenting the UI

information. Trouble arises when too much color is used. If several pieces are strongly colored, it loses its potency to grab the attention. Here the usage of color will be limited to one major and one minor highlight color.

- **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 a 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

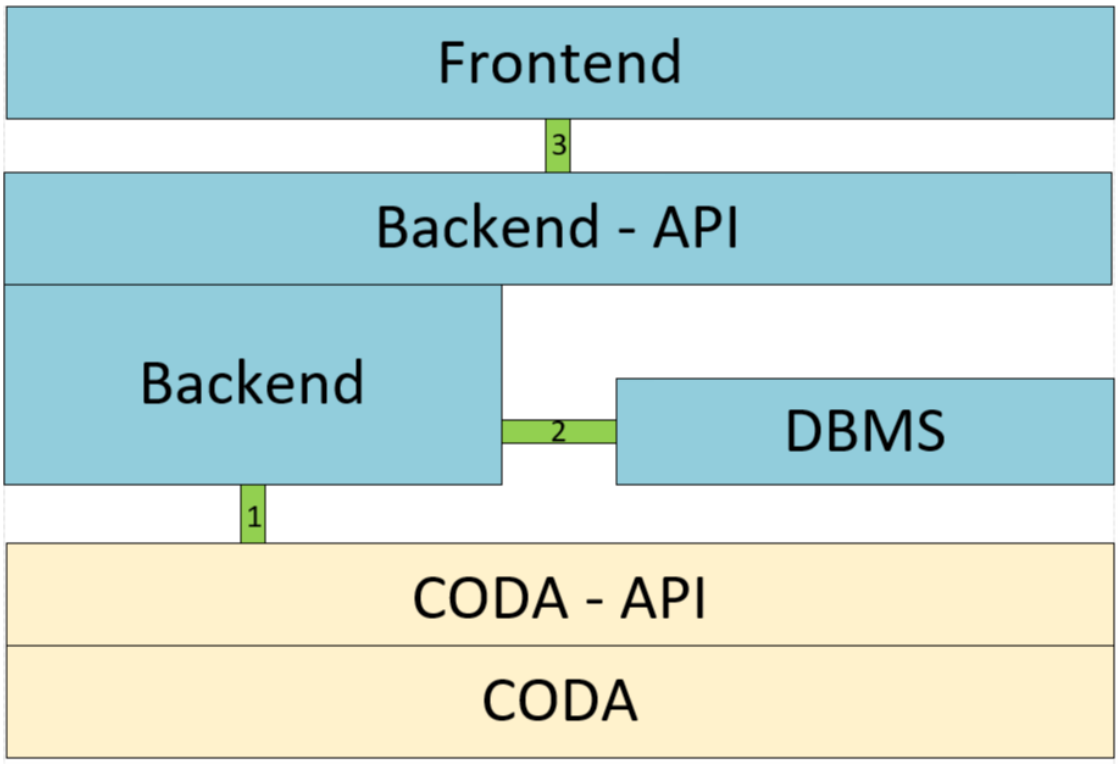

Figure 4.2: 30000-foot view of Sparked architecture

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 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 get a first overview, it is helpful to look at 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.

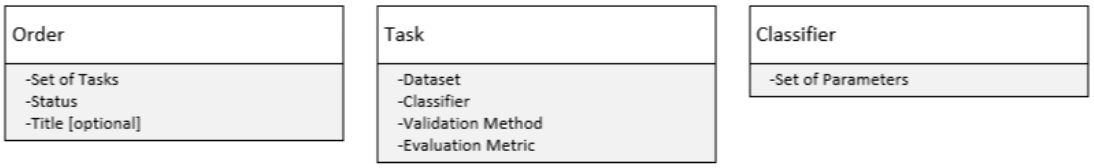

Figure 4.3: Data needed to define an Order

Four endpoints are delivering the supported classifiers, validation methods, datasets, and evaluation metrics. They are supplemented with an endpoint with the status of all Orders, one to get the status of a specific Order, one to get the results for a specific Order and an endpoint each to start a Task or an Order. For the full API see 7

It is to note that CODA is still in development. Therefore the API might be subject to change in the future.

4.2.2 Workflows

Having defined the central building blocks, it is now time to look at the workflows needed to satisfy the functional requirements. To recap, the functional requirements for Sparked are:

- Create an Order
- Start an Order
- List existing Orders
- Display the evaluation data of a completed Order

4.2.2.1 Workflow: Create and start an order

An Order is a set of Tasks, that only differs in the classifier, while a Task combines all information needed to run an evaluation. That an Order may not contain Tasks differing in features other than the classifier is not inherent to the concept of an Order. As such, this boundary condition will not be put into the basic structure, making it easier to change in the future.

The final feature of an Order is status information, reflecting whether the Order is new, has been started, completed or if an error has occurred.

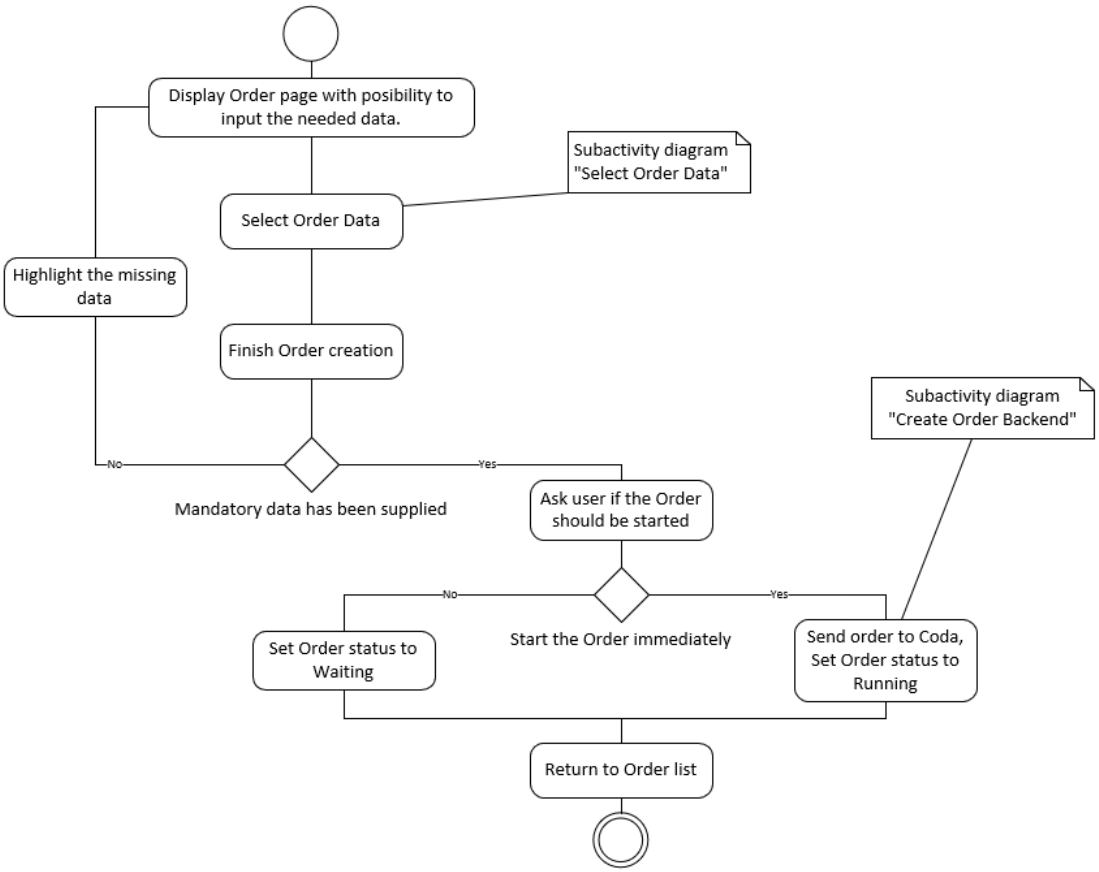

Figure 4.4: Activity Diagram – Create Order

An activity diagram of the Order creation workflow for a better overview. Creating it helps to get into the mindset of what the workflow entails. Later in development, it can be used as a red line on what needs to be done.

From the activity diagram, four parts can be identified.

- 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

An important Sparked usage scenario is to show example Orders while in a presentation. Creating and starting an Order from scratch in such a situation has the problem, that the runtime for ML Tasks can be quite substantial. The user may want to create a fitting Order beforehand and mark it for a special occasion.

Since Sparked has its own data storage, it is possible to add additional data to an Order 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 concepts like labeling or user accounts became relevant, they could be achieved similarly to the title.

4.2.2.2 Workflow: Listing orders

Once an Order can be created, listing and opening all existing Orders becomes the next workflow to consider.

Its main goals of this workflow are:

- See all Orders created by this instance of Sparked
- Show enough information that the user can decide which to open
- Open detail page of a single Order

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 point pertains the Order status. The status is subject to change and its current value can only be supplied by the CODA backend, requiring the UI to 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 towards Sparked whenever a status has changed. The CODA API does not allow for this though, making a polling approach the option of choice.

The data will contain information on all Orders, which makes this a potentially large and costly request. At the same time, the information should not become too old as the user does not want to wait for an Order that has already been finished. As a good polling interval is subjective, this is a value that 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 evaluation workflow has two main functions. Giving the user the details of the open Order and show them the evaluation results that CODA supplies. Of course, showing results is only possible for Orders that have successfully been evaluated by CODA. Opening the Order details from the Order list should however be possible regardless. An Order that has not been started, should be able to be started from here.

The complex part of this workflow lies within the visualization of the evaluation results. At the very least the results for the target metric should be displayed.

Having an unknown number of Classifier makes this even more complex, as the UI must be created dynamically to allow for this.

Chapter 5

Implementation

5.1 Technologies

With a clear picture of the requirements and the aspired architecture, it is time to talk about technology, which 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 [3] [4] 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. Then JavaScript is used to make server calls, animate objects, 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 know-how existed.

Angular is maintained by Google, was first released in 2009 [5] and is available under a modified MIT license [6], with the newest Version being 8.0.0, released on May 28th, 2019 [7]. Sparked uses Version 7.2.3 as it was the newest at development time.

Angular uses TypeScript as a 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 [8] and was first released by Microsoft in 2012 and exists now in Version 3.4 [9]. 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 [10], can be used with the node package manager and has an active community behind it [11]. 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 has libraries, that allows creating a simple CRUD web server. 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 safe bet as it is one of the most widely used programming languages [12].

When it comes to creating web application backends with java Spring Boot, here used in version 2.1.1 is a widely used and powerful tool that will reduce the configuration overhead and supplies powerful tools to the development project.

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 schemaless database like MongoDB is, that changes in the data do not need to be consciously reflected in the database. This makes changes to Sparked easier, as it reduces the points in code that need to be changed if the CODA API were 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 them 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.

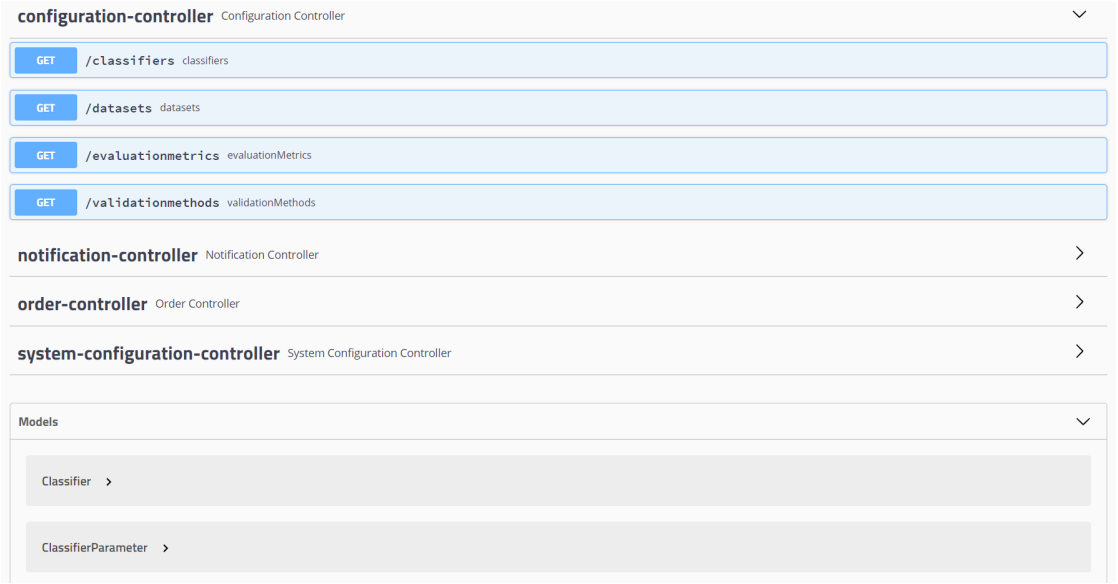

Figure 5.1: The Swagger UI shows a synopsis of Sparkeds web-API

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 and shows supported parameters, responses that can be expected and the data structures of return values. This information should enable frontend developers to change the UI or even create a new one without having to go into the code themselves.

5.2 Noteable code

5.2.1 Coda API

The ML functionality of Sparked comes fully from CODA via the CODA API 7. 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  },
```

Listing 5.1: Metric returned by the CODA API in Json format.

```
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;
```

Listing 5.2: Metric class in Sparked. The fields match the Json fields.

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 deserializer.

```
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",
```

Listing 5.3: Objects with changing variable names need a more complex deserialization procedure.

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. If the evaluation result data were changed, both the matching class as well as the deserializer, EvaluationResultMetricDeserializer, would need to 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.

5.2.2 Configuration

The configurable data in Sparked concerns three issues. The first issue is the MongoDB connection, containing IP-address, port, database name, and collection name as well as timeout values. Another property in this block is the clear_database_on_startup property. If set to true, it will drop the MongoDB collection, deleting all saved Orders. This effectively clears all existing Orders from the program on startup.

The second is the CODA connection, containing the APIs IP-address and port value as well as the polling interval in which to query the backend for changed status information as described in the Order list workflow description.

```
1  mongo_Port=8080
2  mongo_IP=40.68.62.58
3  mongo_Database_Name=coda_db
4  mongo_Collection_Name=orders
5  mongo_Connection_Timeout=1000
6  mongo_Server_Selection_Timeout=0
7  mongo_Socket_Timeout=1000
8  clear_database_on_startup=false
9
10 order_Status_Intervall=60
11 coda_backend_ip=10.0.2.55:5000/
```

Listing 5.4: The config.properties file.

The properties are loaded with the help of the Properties class from java.util. This is encapsulated into a custom properties class, which has been configured to support dependency injection. The third property block concerns the logger. In Sparked a very common logging framework, log4j is used. Log4j allows its configuration values like log level, data sinks, and the statement pattern to be configured via a config file. Log4js configuration file, log4j2-spring.xml, can be found in the resources folder. 7

5.2.3 Dependency Injection

Spring Boot brings the capability to inject dependencies directly to where they are needed. In the package coda.di the dependency injection capable classes are configured. Within a normal class injected fields can be recognized by the @Autowired attribute

before the variable definition. This is most commonly used to inject the logger and the properties classes.

These injections are done against interfaces, to help decouple the application as far as possible.

5.2.4 Some more code

There are a couple of other interesting points in the code that deserve a short mentioning.

With the use of filters in the interceptor package, all exceptions are caught, there should be no situation, where internal exception data is published to the frontend. As a consequence, the frontend has no way of knowing if a server-side exception has occurred.

A second filter is set up to log every request to Sparked if log level trace is configured.

In the frontend, Sass is used to work with variables. The variable.scss file defines is supposed to be the central spot to define all styling information, that is used in different parts of the UI.

Chapter 6

Evaluation

In chapter three the requirements for Sparked have been defined via a couple of functional requirements that lead to four workflows:

- Create an Order
- Start an Order
- List existing Orders
- Display the evaluation data of a completed Order

In the same chapter, a handful of guidelines had been developed after which to design the UI.

- Reduce the information density
- High contrast, big font
- Use a strong color to guide the attention
- Use the same look and feel throughout the application

While creating Sparked, all four of these workflows found their spot in the application.

Configuring and starting an Order works. After sending an Order to CODA for evaluation, the listAllEvaluationStatus endpoint shows the Orders combined with a status courtesy of the CODA backend.

Opening an Order shows its evaluation page. If the status is completed, the result data is loaded from CODA and displayed together with the general information. The

Figure 6.1: An example of the Create Order workflow.

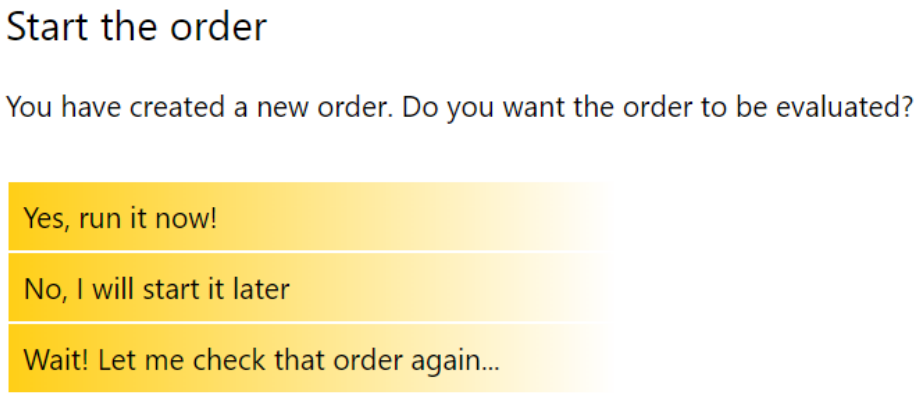

Figure 6.2: Sparked asking for confirmation when finishing an Order.

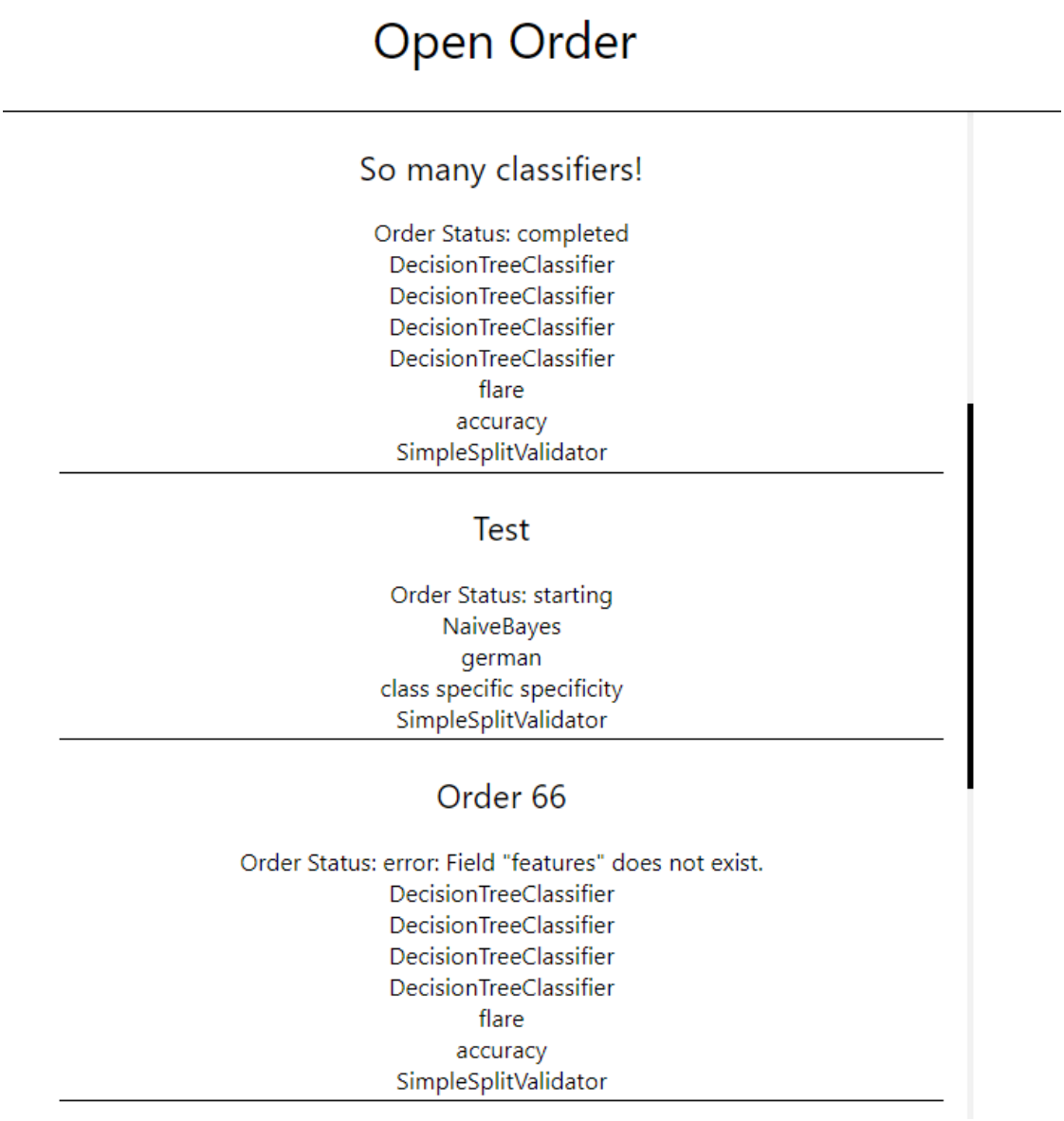

Figure 6.3: List of Orders. The Order Status is supplied by CODA, showing that a connection exists.

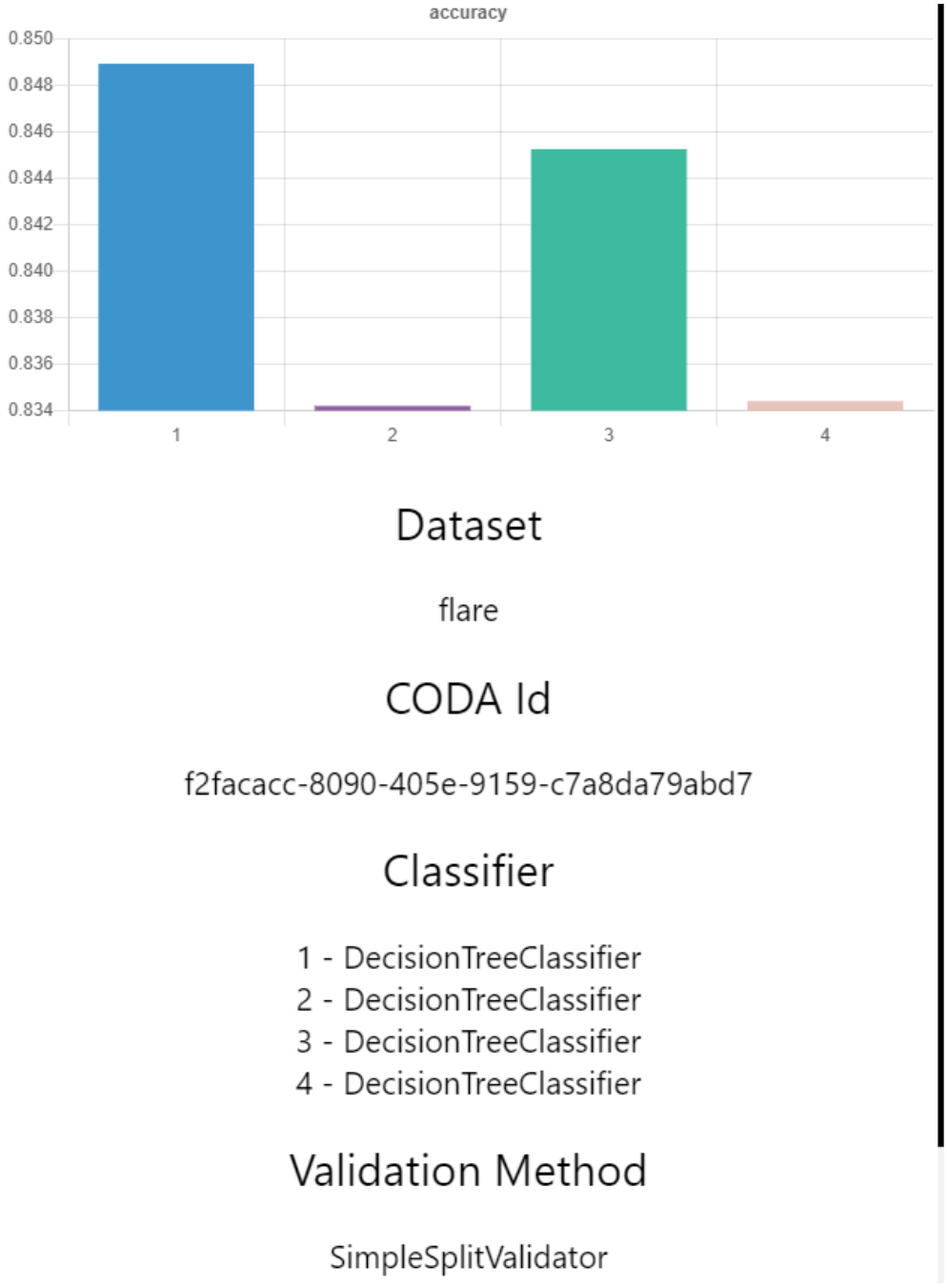

Figure 6.4: An Order created via Sparked with visualized result data. All workflows have to work to get to this point.

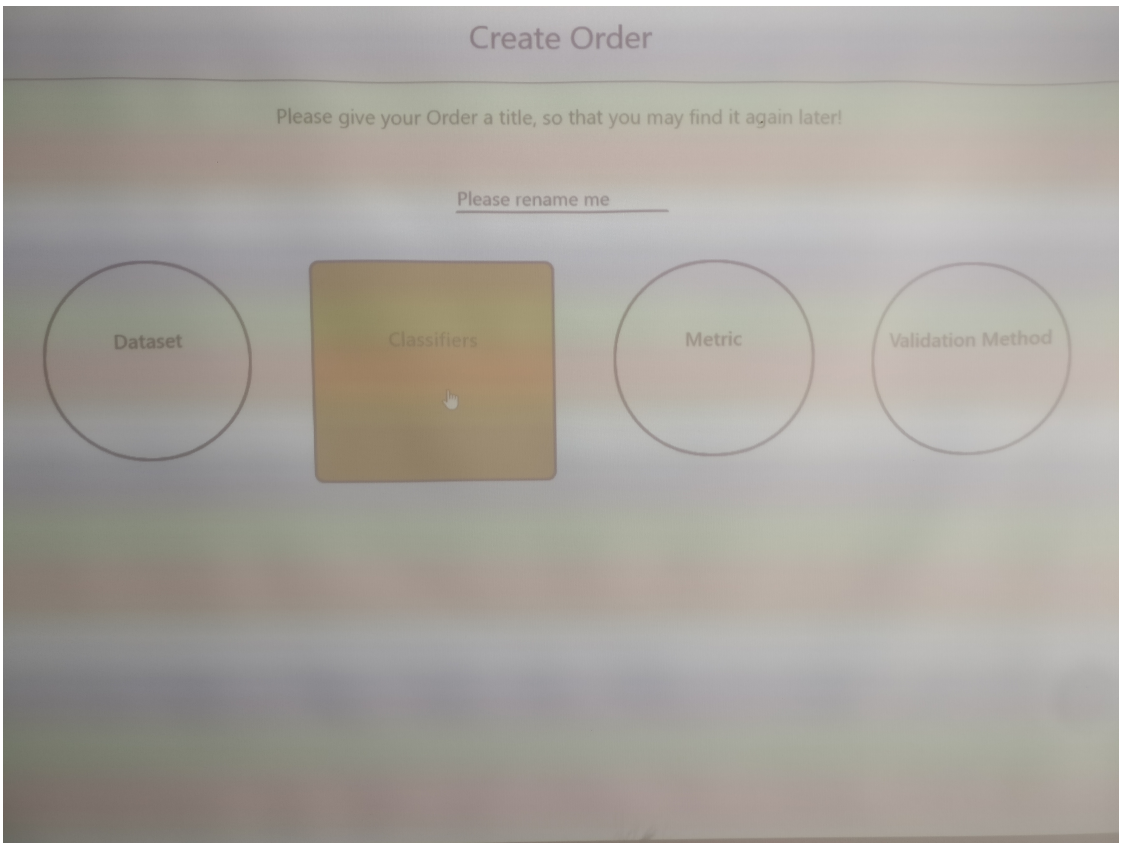

Figure 6.5: The Sparked UI on a beamer in a bright room with indirect but strong sunlight photographed with a smartphone camera.

evaluation results are displayed in a bar chart. At the moment this will only display the measurement of the defined evaluation metric.

Next to others an Order with the flare dataset, four decision tree classifiers with different hyperparameters, accuracy as evaluation method and a simple split validator was created and subsequentially started. The order was successfully evaluated and the data shown on the evaluation page (IMAGE X+Z), showing that all parts work in tandem.

In an attempt to test readability of the Sparked UI under bad lighting conditions the UI was projected onto a screen while having strong if indirect sunlight (Image X+Y). As expected, the UI looks washed out but is still easily readable.

Chapter 7

Conclusion and future development

Creating Sparked became an interesting experience in knowing technology versus having mastered it. Being able to check off 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 to do so 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.

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 has only been used to a small degree. Here the code quality would have benefited even more from a real frontend developer with Angular experience.

That said, the frontend 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 be simple to use and good to read.

As can be seen in the previous chapter, the functional requirements have in principle been met. But the evaluation was quite shallow, only testing with a small handful of test Orders. Showing only the results for the target metric, too, should be expanded on.

Even more problematic is the evaluation of the non-functional requirements. The visual guidelines have in principle be followed, but a qualitative evaluation of the UI was not done. There are a couple of areas, where future development could help Sparked become a more useful application.

The UI would probably benefit a lot from a domain expert going over the different views and deciding what information the user would need. 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 knowledge should allow for easy changes.

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

Last but not least the evaluation page should be extended. Since all relevant data is now available in the evaluation workflow, it should be possible to extend the evaluation page without changing anything else, reducing the amount of work needed to a minimum.

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] LONG, FRANK: *Real or imaginary: The effectiveness of using personas in product design*. In *Proceedings of the Irish Ergonomics Society Annual Conference*, volume 14, pages 1–10. Irish Ergonomics Society Dublin, 2009. 10

[3] MICROSOFT: *Microsoft product livecycle*. https://support.microsoft.com/en-us/lifecycle/search?alpha=Silverlight%205. 21

[4] COMMUNICATIONS, ADOBE CORPORATE: *Flash & The Future of Interactive Content*. https://theblog.adobe.com/adobe-flash-update/. 21

[5] WIKIPEDIA: *AngularJS*. https://de.wikipedia.org/wiki/AngularJS. 22

[6] ANGULAR: *Angular*. https://angular.io/. 22

[7] ANGULAR: *Releases / Angular*. https://github.com/angular/angular/releases. 22

[8] TYPESCRIPT: *TypeScript/License.txt*. https://github.com/microsoft/TypeScript/blob/master/LICENS
22

[9] TYPESCRIPTLANG.ORG: *TypeScript - JavaScript that scales.* https://www.typescriptlang.org/. 22

[10] CHART.JS: *Chart.js*. https://github.com/chartjs/Chart.js. 22

[11] GITHUB: *contributors to chartjs*. https://github.com/chartjs/Chart.js/graphs/contributors. 22

[12] STACKOVERFLOW: *Developer Survey Results 2019*. https://insights.stackoverflow.com/survey/2019. 23

List of Figures

List of Tables

Abkürzungsverzeichnis

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

Appendix

CODA API

Log4j configuration file

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration>
  <Appenders>
    <File name="fileAppender" fileName="logs/log.log">
        <PatternLayout pattern="%d{yyyy-MM-dd HH:mm:ss.SSS} %-5level - %msg%n"/>
    </File>
</Appenders>

  <Loggers>
    <Root level="warn">
    </Root>

    <Logger name="coda.shared" level="debug">
        <AppenderRef ref="fileAppender" />
    </Logger>
  </Loggers>
</Configuration>
```

Listing 1: Log4j configuration file.

API	Description	Parameters	Return type	URL
GET /evaluation/listClassifiers	Returns the list of the classifiers in JSON format	none	String	http://10.0.2.55:5000/evaluation/listClassifiers
GET /evaluation/listValidationMethods	Returns the list of validation methods in JSON format	none	String	http://10.0.2.55:5000/evaluation/listValidationMethods
GET /evaluation/listEvaluationMetrics	Returns the list of evaluation metrics in JSON format	none	String	http://10.0.2.55:5000/evaluation/listEvaluationMetrics
GET /evaluation/listDatasets	Returns the list of the dataset in JSON format	forcedRefresh: boolean forcedRefresh=true then, it retrieve the data from the repository, if forcedRefresh=false, it retrieves the data from the cache	String	http://10.0.2.55:5000/evaluation/listDatasets?forcedRefresh=true
GET /evaluation/listAllEvaluationStatus	Returns the list of the evaluation id along with their status in JSON format	none	String	http://10.0.2.55:5000/evaluation/listAllEvaluationStatus
GET /evaluation/getStatus	Returns the Status of the evaluation id. It returns whether the API has been completed,cancelled, or has occured and exception. If an exception occurs, it displays the probable cause for the exception. For eg, if the user does not mention the validationMethod name, then it will throw and exception stating "Please provide validation method".	id : String	String	http://10.0.2.55:5000/evaluation/getStatus
GET /evaluation/getResults	Returns the Results of the evaluation id in JSON format	id : String	String	http://10.0.2.55:5000/evaluation/getResults

Figure 1: CODA GET endpoints

API	Description	Parameters	Return Type	URL
POST /evaluation/launchClassifier	Returns the id of the evaluation,in the background it launches evaluation	datasetId:string, classifierName:obj, hyperParams:obj, validationMethod:obj, evaluationMetric:String	String	http://10.0.2.55:5000/evaluation/launchClassifier
POST /evaluation/launchEvaluation	Returns the id of the evaluation, in the background it launches evaluation for common validation method and evaluation metrics	datasetId:string, classifiers:obj, validationMethod:String, evaluationMetric:String	String	http://10.0.2.55:5000/evaluation/launchClassifier

Figure 2: CODA POST endpoints