University of Zurich[UZH]

BACHELOR THESIS – Communication Systems Group, Prof. Dr. Burkhard Stiller

Creation of a Platform to Compute the Trustworthiness Level of Unsupervised and Supervised ML/DL Models

Said Haji Abukar
Student ID: 16-724-718

Supervisor: Dr. Alberto Huertas Celdran, Muriel Franco
Date of Submission: March 25, 2023

University of Zurich
Department of Informatics (IFI)
Binzmuehlestrasse 14, CH-8050 Zurich, Switzerland

Bachelor Thesis
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmuehlestrasse 14, CH-8050 Zurich, Switzerland
URL: http://www.csg.uzh.ch/

Abstract

AI has the potential to revolutionize industries and improve daily life through the development of advanced maschine learning (ML) and deep learning (DL) models. These models, such as chatbots and language models, use algorithms or artificial neural networks to recognize patterns and make decisions. ML involves training algorithms on large datasets to identify patterns and make decisions, while DL uses artificial neural networks composed of interconnected nodes called artificial neurons to process and transmit information. Neural networks can learn and make decisions by adjusting the connections between neurons based on input data. There are two types of ML and DL: unsupervised and supervised. Unsupervised learning involves using algorithms or neural networks to learn from data without labeled outcomes, while supervised learning involves training algorithms or neural networks on labeled data to make predictions or decisions. As AI becomes more advanced and widespread, it is important to have confidence in the decisions and actions of these systems. Trusted AI refers to the reliability and ethical behavior of AI systems. It is crucial to have a framework for evaluating the trustworthiness of different AI models to ensure their safe and responsible deployment. A taxonomy of pillars and metrics can be used to quantify the trustworthiness of AI models, allowing for a structured and comprehensive evaluation of their strengths and limitations. The following bachelor thesis aims to survey existing platforms, define requirements and develop a web app that allows the computation of the trustscore, pillarscores, metricscores of supervised and unsupervised and DL atform is extended to allow for user management, and the return of the trustworthiness levels via API endpoints.

Zusammenfassung

KI hat das Potenzial, Branchen zu revolutionieren und das taegliche Leben durch die Entwicklung fortschrittlicher ML- und DL-Modelle zu verbessern. Diese Modelle, wie z. B. Chatbots und Sprachmodelle, verwenden Algorithmen oder kuenstliche neuronale Netze, um Muster zu erkennen und Entscheidungen zu treffen. Bei ML werden Algorithmen auf grossen Datensaetzen trainiert, um Muster zu erkennen und Entscheidungen zu treffen, waehrend DL kuenstliche neuronale Netze verwendet, die aus miteinander verbundenen Knoten, den sogenannten kuenstlichen Neuronen, bestehen, um Informationen zu verarbeiten und zu uebertragen. Neuronale Netze koennen lernen und Entscheidungen treffen, indem sie die Verbindungen zwischen Neuronen auf der Grundlage von Eingabedaten anpassen. Es gibt zwei Arten von ML und DL: unueberwachtes und ueberwachtes Lernen. Beim unueberwachten Lernen werden Algorithmen oder neuronale Netze verwendet, um aus Daten ohne gekennzeichnete Ergebnisse zu lernen, waehrend beim ueberwachten Lernen Algorithmen oder neuronale Netze auf gekennzeichneten Daten trainiert werden, um Vorhersagen oder Entscheidungen zu treffen. Da die KI immer weiter fortgeschritten und verbreitet ist, ist es wichtig, dass man sich auf die Entscheidungen und Handlungen dieser Systeme verlassen kann. Vertrauenswuerdige KI bezieht sich auf die Zuverlaessigkeit und das ethische Verhalten von KI-Systemen. Es ist von entscheidender Bedeutung, einen Rahmen fuer die Bewertung der Vertrauenswuerdigkeit verschiedener KI-Modelle zu haben, um ihren sicheren und verantwortungsvollen Einsatz zu gewaehrleisten. Eine Taxonomie von Saeulen und Metriken kann verwendet werden, um die Vertrauenswuerdigkeit von KI-Modellen zu quantifizieren und eine strukturierte und umfassende Bewertung ihrer Staerken und Grenzen zu ermoeglichen.

Ziel der folgenden Bachelorarbeit ist es, eine Webapplikation zu entwickeln, die die Berechnung des Trustscores, Pillarscores und Metricscores von ueberwachten und unueberwachten Machine Learning und Deep Learning Modulen (ML & DL) ermoeglicht. Eine bestehende Plattform wird erweitert, um die Benutzerverwaltung und die Rueckgabe der Scores ueber API-Endpunkte zu ermoeglichen.

Acknowledgments

I would like to thank the Communication Systems Research Group at UZH for their help and support during the project.

My special thanks to Dr. Alberto Huertas and Prof. Dr. Burkhard Stiller for making it possible for me to work on this project.

Contents

Chapter 1

Introduction

1.1 Motivation

Artificial intelligence (AI) has revolutionized many aspects of our lives. For example, personal assistant AI systems, such as Siri, enable the accomplishment of tasks through natural language commands, relying mainly on supervised machine learning (ML) where an algorithm learns from labeled data to make predictions about new, unlabeled data [2]. In finance, FICO's Falcon Fraud Manager uses unsupervised ML for credit card fraud detection to identify patterns of fraudulent behavior. The algorithm learns from unlabeled data to discover patterns or relationships in the data to detect fraudulent activities [13]. In natural language processing (NLP), OpenAI's GPT (Generative Pre-trained Transformer) model is a tool that uses unsupervised deep learning (DL) to generate responses to user queries in chatbots. The neural network is trained on unlabeled data to discover patterns or structure in the data, enabling the chatbot to generate appropriate responses [14].

The increasing availability of data and advances in ML and DL algorithms have led to a surge in the use of AI for various applications. Supervised ML involves learning from labeled data to make predictions about new, unlabeled data, while unsupervised ML involves learning from unlabeled data to discover patterns or relationships in the data. Supervised DL involves training a neural network using labeled data to make predictions, while unsupervised DL involves training a neural network on unlabeled data to discover patterns or structure in the data. The use of AI in various domains has increased the efficiency of many processes, leading to improved decision-making and better outcomes for businesses and individuals. However, the trustworthiness of AI systems is becoming an increasingly important issue. In human contexts, we often trust individuals based on their track record, credentials, and other factors that can assure us of their competence and integrity. For instance, we trust a surgeon because they have undergone years of rigorous training, have extensive experience, and possess a license that ensures they are qualified to operate. Similarly, we may trust a financial advisor if they have a good track record of providing sound investment advice and have professional certifications.

In the same way, we need to ensure that the AI systems we use are trustworthy. With regards to supervised ML, we need to be confident that the model is making accurate

predictions and is not biased in favor of certain groups or outcomes. For instance, a loan approval system that systematically rejects applications from a particular race or gender is not trustworthy and could lead to serious consequences for the affected individuals. Similarly, unsupervised ML and DL systems can also pose significant risks if they are not trustworthy. For example, a self-driving car that is not robust and cannot detect pedestrians or obstacles may cause accidents and endanger lives.

Trustworthiness can be defined as the degree to which an AI system can be relied upon to achieve its intended objectives in a safe and ethical manner. Recent literature has identified three perspectives for evaluating the risk and trustworthiness of AI, focusing on technical aspects, the user experience, and the social/policy dimension [5]. Trust in AI for users is based on their perception and mental model of the system, with a trustworthy AI system being available, easy to use, and privacy-protective. From a technical perspective, the system should be accurate, robust against adversarial attacks, and transparent to developers. For policy makers and the social dimension, compliance with regulations and ethical principles is required, with clear responsibility and accountability for the system's actions and decisions. These principles, defined by various stakeholders including the EU, OECD, NIST, and trusted AI committees, emphasize the importance of fairness, explainability, robustness, and accountability as the pillars for trusted AI.

Those four pillars of trustworthiness provide a useful framework for assessing the trustworthiness of AI systems.

Fairness is concerned with how the model treats individuals or groups, and its avoidance of discrimination. Bias is a major issue related to fairness, which can be introduced by human prejudice in the training dataset, assumptions made during the process of training machine and deep learning models, or incomplete or not diverse enough training datasets. To measure bias and ensure fairness, a common metric is the "disparate impact ratio," which measures the ratio of the outcomes for different groups (e.g. protected vs. non-protected groups) and compares them to the expected outcomes [6]. A disparate impact ratio greater than one indicates that one group is being unfairly advantaged, while a ratio less than one indicates unfair disadvantage [6] Other Metrics for assessing fairness include underfitting and overfitting, which compute the difference between train and baseline or train and test performance, and class balance, which measures the ratio of samples in different training set classes as well as statistical parity, equal opportunity and average odd difference.

Explainability refers to the transparency and understandability of the model's decision-making process. The major issue with explainability is the opacity of many machine learning models, which can make it difficult for stakeholders to understand how the model arrived at a particular decision or prediction. To address this issue, a common metric is "feature relevance," which measures the relative contribution of each input variable to the model's output. Feature importance can be calculated using methods such as permutation importance, which involves randomly permuting each feature and measuring the resulting decrease in the model's performance. The features with the largest decrease are considered the most important [6].

Robustness refers to the model's resistance to adversarial attacks or manipulation. The major issue with robustness is the model's susceptibility to adversarial attacks, which refer

to deliberate attempts to manipulate the model's inputs in order to produce incorrect or malicious outputs. To measure robustness and ensure that the model can resist adversarial attacks, a common metric is the "confidence score," which measures the model's degree of certainty in its predictions [6]. A low confidence score indicates that the model is uncertain and may be more susceptible to adversarial attacks, while a high confidence score indicates that the model is confident in its prediction [6].

Accountability refers to the model's ability to be held responsible for its actions and consequences particularly in cases where the model produces unintended or harmful outcomes. It is often combined with transparency aspects in respect to its stakeholders. To measure accountability and ensure that the model can be held responsible for its actions, a common metric is "train-test split," which involves dividing the data into training and testing sets and measuring the model's performance on both. If the model performs well on the training set but poorly on the testing set, it may indicate that the model is overfitting to the training data and may not generalize well to new data [6]. A poor performance on the testing set may indicate a lack of accountability on the part of the model [6].

There exist several limitations in the current state of tools and platforms used for assessing the trustworthiness of supervised and unsupervised machine learning (ML) and deep learning (DL) models. Current toolkits only evaluate a portion of the factors that contribute to model trustworthiness and often lack user-friendly explanations for their evaluations, while also being inflexible in adapting to specific datasets or tasks. Additionally, the focus on toolkits, rather than standalone software, restricts the usability of the platform.

The current platform has several limitations, such as the absence of user management, APIs, and the ability to create scenarios and solutions in real-time. To address these limitations, this thesis seeks to develop a web application that provides those elements.

1.2 Description of Work

This Bachelor thesis will design and develop a web application to evaluate the trustworthiness of supervised and unsupervised ML/DL models. The process will involve conducting a survey of existing trusted AI platforms, identifying the requirements and use cases for the proposed platform, analyzing suitable web technologies, and formally describing the design. The web application will provide a user interface for uploading ML models and their training data, which will trigger the calculation of trust scores and visualization of the results through graphs. The web application will also generate a report with detailed insights on the trustworthiness of the model. The web application will be validated in a specific application scenario, in which it will be tested and evaluated based on certain criteria.

1.3 Thesis Outline

The work is divided into seven chapters. Chapter 2 deals extensively with existing tools for calculating the trustworthiness of unsupervised and supervised ML/DL models. This chapter has a research character and paves the way for the practical part of this work: the implementation and extension of a platform for calculating the trustworthiness of unsupervised and supervised ML/DL models. Chapter 3 takes a closer look at the webapp requirements, functionality and web technologies, compares them and makes a choice based on front- and backend technologies. The design and implementation are described in chapters 4 and 5. Chapter 6 evaluates the web app based on the requirements. Finally, in Chapter 7, a summary and conclusion of the bachelor's thesis is given. In addition, limitations and future work will be discussed.

Chapter 2

Related Work

2.1 Trusted AI Platforms for ML/DL

Besides the platform proposed by the institute, there is none that combines the scores for different pillars to compute a global trustworthiness score. FAT Forensics is an open-source Python package designed to address shortcomings in the field of AI and ML Fairness, Accountability, and Transparency (FAT) [12]. Hereby accountability is composes of robustness, safety, security & privacy and transparency includes interpretability & explainability, diverging from the four pillar taxonomy. It provides a common application programming interface for a range of algorithms and is designed to be compatible with popular machine learning toolkits such as scikit-learn. The toolbox can be used in two modes: research mode, which is intended for use in interactive environments such as Jupyter notebooks and allows for prototyping and exploratory analysis; and deployment mode, which can be used as part of a data processing pipeline to provide numerical FAT analytics and support automated reporting and dashboarding. Following for three pillars differnt tools are presented:

2.1.1 Fairness

Table 2.1 summarizes open-source toolkits and frameworks for fairness in AI. One of the tools is introduced below.

FairTest is a toolkit designed to detect and debug unwarranted associations in data-driven applications, which are strong associations between the outputs of an algorithm and features defining a protected user group that arise in a meaningful subset of users and have no explanatory factors. The toolkit offers a systematic methodology for testing for and debugging such associations through five steps, including data collection and identification of user features and algorithmic outputs of interest, integration of explanatory factors, mapping of the data and application of an appropriate statistical metric, testing for association bugs over semantically meaningful user subpopulations, and debugging and fairness evaluation.

The toolkit supports three core investigative primitives, including the discovery of association bugs, testing for suspected association bugs, and error profiling of an ML algorithm over a user population. FairTest also employs a novel technique called association-guided tree construction to efficiently identify semantically meaningful subpopulations affected by association bugs.

FairTest considers three types of user attributes, including protected attributes, which are discrimination-sensitive features, contextual attributes, which are dimensions along which the user population can be split into semantically meaningful subpopulations, and explanatory attributes, which are user properties that are acceptable to differentiate, even if that leads to apparent discrimination on protected attributes.

The toolkit uses a set of five canonical metrics to measure the strength or extent of the association between algorithm outputs and protected user attributes. FairTest tests for significant associations and, if found, selects a model that explains the association. The model should be interpretable and satisfy any relevant constraints, and the selected model is evaluated for fairness violations and debugging purposes. The FairTest API includes the core Investigation class, subclassed in the three investigation types. A developer first collects user attributes and applications outputs as DataSource and can then derive context for one or more investigations with a train function, test correct associations with the test method, and return a report with the report method. The API also includes a way to implement metrics.[1]

Toolkit	Characteristsic
FairTest	Analyzing associations between outcomes and sensitive attributes.
Themis, Aequitas, ExpGA	Black-box random discriminatory instance generation
fairCheck, MLCheck	Verification-based discriminatory instance generation
LTDD, Fair-SMOTE	Detecting which data features and which parts of them are biased
xFAIR	Extrapolation of correlations among data features that might cause bias
Fairway	Detecting biased data labels and optimal hyper-parameters for fairness
Parfait-ML	Searching for hyper-parameters optimal to ML software fairness
Fairea	Testing fairness repair algorithms
AIF360, scikit-fairness, LiFT	Examining and mitigating discrimination and bias in ML software
FairTest	Measuring bias that occurs in each stage of the ML life cycle
FairVis	Visual analytics for discovering intersectional bias in ML software
BiasAmp	Analyzing whether ML exacerbates bias from the training data
FairRepair	Fairness testing and repair for tree-based models
SBFT	Search-based fairness testing for regression-based ML systems
ADF, EIDIG, NeuronFair	White-box search-based discriminatory instance generation for DNNs
CMA, FairNeuron	Detecting which parts of DNNs are responsible for unfairness
ASTRAEA	Grammar-based discriminatory instance generation for NLP systems
MT-NLP, BiasFinder	Template-based discriminatory instance generation for NLP systems
REVISE	Detecting object-, gender-, and geography-based bias in CV datasets
FINS	Group fairness testing for subset selection tasks

Table 2.1: Open source tools for fairness testing.[7]

2.1.2 Explainability

Table 2.2 summarizes open-source toolkits and frameworks for explainability in AI. One of the tools is introduced below. The AIX360 toolkit provides an easy-to-use programming interface and flexible software architecture to support a range of explainability techniques required by different stakeholders. The programming interface is similar to popular Python model development tools, making it accessible to data scientists who may not be experts in explainability. Additionally, a hierarchy of Python classes corresponds to explainers for data, models, and predictions, with base class explainers organized according to the AI modeling pipeline. The toolkit includes various explainer classes, such as Data Explainers, Directly Interpretable Explainers, Local Post-Hoc Explainers (both black-box and white-box), and Global Post-Hoc Explainers (also both black-box and white-box). AIX360 also includes dataset classes and framework-specific classes for Keras and PyTorch models, allowing explainability algorithm developers to avoid implementing algorithms multiple times for each framework. The toolkit is designed to be flexible and extensible, enabling developers to integrate new explainability algorithms with ease. A code example is provided in below, demonstrating the use of the BRCGExplainer, a directly interpretable explainer [4].

```
from aix360.algorithms.rbm import
BRCGExplainer, BooleanRuleCG
# Instantiate and train an explainer to
compute global rules in conjunctive
normal form (CNF)
br = BRCGExplainer(BooleanRuleCG(CNF=
True))
br.fit(x_train, y_train)
# print the CNF rules
print (br.explain()['rules']
```

[4]

Toolkit	Characteristsic
InterpretML	Microsoft-developed solution that provides both white-box models (e.g., decision trees, rule lists) and model-agnostic methods for explaining black-box models. The methods in InterpretML are designed specifically for tabular data.
AIX360	Microsoft-developed solution that provides both white-box models (e.g., decision trees, rule lists) and model-agnostic methods for explaining black-box models. The methods in InterpretML are designed specifically for tabular data.
Skater	Oracle-developed solution, has been around since 2017 and provides model-agnostic procedures for local and global explainability. Additionally, Skater includes two local explanatory methods for deep neural networks
Alibi	Developed by Seldon Technologies Ltd, a company specializing in AI and DevOps. Similar to AIX360, alibi provides various model-agnostic and model-specific methods for local and global explainability.
iNNvestigate, DeepExplain	Two librariers developed by researchers that offer a range of explanation methods.
Tf-explain	developed by French DeepTech startup Sicara, is another framework that specializes in explaining deep neural networks.
Captum	is a neural network explanation library that is specifically designed for use with PyTorch. It is currently being developed by Facebook.
SHAP	is the implementation of the explanation method of the same name. The authors provide this themselves.
LIME	is the implementation of the explanation method of the same name. The authors provide this themselves.

Table 2.2: Open source tools for explainability testing.[4]

2.1.3 Robustness

Table 2.3 summarizes open-source toolkits and frameworks for Robustness in AI. One of the tools is introduced below.

The Adversarial Robustness Toolbox (ART) is a Python toolkit developed by IBM to improve the robustness of machine learning models against external attacks. It is an open source toolkit that provides various features and mechanisms to detect and defend against attacks by attackers. The toolkit is based on a well-structured and modular framework that allows researchers and developers to experiment with different techniques to make machine learning models more resistant to attacks[19].

ART includes several implementations of attacks and defenses that can be used to evaluate the robustness of machine learning models. It also provides several tools for measuring the robustness of models, such as the CLEVER score, loss sensitivity, empirical robustness, and the Clique method[19].

The CLEVER score is used to measure the sensitivity of a model to adverse disturbances, while loss sensitivity measures the impact of small changes in input on the model's output. Empirical robustness measures the minimum perturbation required to change the model's prediction, and the clique method is used to find the minimum perturbation from the adversary[6].

The core functionality of ART focuses on improving the robustness of machine learning models through various attack detection and defense strategies [3]. These strategies are designed to detect and prevent attacks before they can cause damage. ART includes several defense strategies, such as the adversarial training method, in which the model is trained with adversarial examples, and the feature scattering method, in which random perturbations are added to the input features.

ART also includes a data generator for creating counterexamples that can be used to test and validate the effectiveness of attack detection and defense strategies. It provides several visualization tools that allow users to visualize the impact of adversarial attacks on their machine learning models.

Toolkit	Characteristsic
Cleverhans	A Tensorflow library for benchmarking the vulnerability of machine learning models to adversarial examples.
Advertorch	A Pytorch toolbox containing attack methods for image classification.
DeepRobust	A Pytorch toolbox containing attack and defense methods in image and graph domain.
RobustBench	A tool to standardize the evaluation of adversarial robustness.
Adversarial Robustness Toolbox	A Python toolkit developed by IBM to improve the robustness of machine learning models against external attacks.

Table 2.3: Open source tools for robustness testing.[4]

It appears that even after intensive literature research, no solution has been found for calculating the methodology score or for providing a global measure of the trustworthiness pillars.

Chapter 3

Requirements

3.1 Trusted-AI Webapp

The main focus of this bachelor thesis was to extend the current web application "Trusted-AI," which was originally created by Joel Leupp, Melike Demirici, and Jan Bauer as a master project and later expanded by Mauro Doerig in a bachelor thesis. The first iteration of the web app allowed for the computation of pillar and metric scores, as well as the trust score for supervised machine learning and deep learning models. The second iteration extended the web app to include unsupervised models.

The web app was structured with an upload module for scenarios, a solution creation/upload module, and an analysis module that allowed for the computation and display of trust and performance scores in the frontend. The backend was composed of the Trusted AI/Trusted Anomaly Detection algorithm, which performed computations based on user input from the frontend modules, and a database for storage.

3.2 Terminology

The created web app adopts and modifies the terminology of scenarios and solutions. A scenario is a Machine Learning (ML) and Deep Learning (DL) task with a given name and a clear description. A model solving such a task is termed a "solution"; it's either an "unsupervised solution" for unsupervised ML and DL models solving the scenario or a "supervised solution" if it's a supervised ML and DL model that solves the scenario.

3.3 Requirements

The requirements for the created web app can be categorized by purpose as described below:

1. User management (creation, deletion, updating of users and an admin)
2. Scenario retrieval creation, update, deletion
3. Supervised/unsupervised solution creation, update, deletion
4. Getting the metric, pillar scores, and trust score of supervised/unsupervised solutions on a graphical user interface (GUI) and via API endpoints

Next to the implementation in the graphical user interface, as API endpoints or relying on framework-specific characteristics, the requirements can be categorized in those concerning:

1. User management all users
2. specific for admin users
3. specific for non-admin users

Moreover, the scenario and solution-related, non-admin specific requirements are:

- It must have a graphical user interface that allows all users to create a scenario by providing a scenario_name and description.
- It must have a graphical user interface and API endpoints that allow all users to view all scenarios created by them
- It must allow admin users to view all scenarios a specific user creates by providing their username.
- It must allow all users to edit the name and/or the description of a scenario they have created through the web app's graphical user interface or an API endpoint.
- It must allow all users to upload an unsupervised or supervised solution to the system by providing solution details such as solution name, solution type, training file, test file, protected features, protected values, target column, favorable outcomes, factsheet file, model file, metrics mappings file, and weight metrics.
- The web app must allow users to edit the fields of an unsupervised/supervised solution they created using its graphical user interface or API endpoints.
- The web app must allow all users to delete unsupervised/supervised solutions they created using its graphical user interface or an API endpoint.

The scenario and solution-related, admin-specific requirements are:

- The web app must allow admin users to delete, get, and update an unsupervised/-supervised solution of a user with given user name and solution name.
- The web app must allow admin users to delete, get, and update a scenario of a user with given user name, scenario name.
- The web app must allow admin users to get all scenarios of a user with given user name.
- The web app must allow admin users to get all supervised / unsupervised solutions of a user with given user name.

Graphical user interface (GUI) specific requirements are

- The GUI must allow users to see the performance metrics, metrics, pillar scores, and trust scores of selected supervised/unsupervised solutions belonging to selected scenarios.
- The GUI must allow users to see the performance metrics, metrics, pillar scores, and trust score of two selected supervised/unsupervised solutions belonging to the same scenario.
- The GUI must show users the metric, pillars and trust score of the last uploaded /analyzed supervised / unsupervised solution.
- The GUI must allow users to register and login into the web app.

3.4 Functionality

- Create Scenario: A user can create a new scenario by providing a scenario name and description via the web app's graphical user interface.
- View All Scenarios: Admins can view all scenarios that all users have created using the web app's graphical user interface or API endpoints.
- View Own Scenarios: Users can view their own scenarios they have created using the web app's graphical user interface or API endpoints.
- Edit Scenario Fields: A user can edit the name and/or the description of a the scenario they have created using the web app's graphical user interface or API endpoints.
- Upload Supervised Model: A user can upload a supervised solution to the system by providing solution details such as solution name, solution type, training file, test file, protected features, protected values, target column, favorable outcomes, fact sheet file, model file, metrics mappings file, and weight metrics via the web app's graphical user interface.

- Upload Unsupervised Model: A user can upload an unsupervised solution to the system by providing solution details such as solution name, solution type, training file, test file, outliers file, protected features, protected values, target column, fact sheet file, model file, metrics mappings file, and weight metrics via the web app's graphical user interface.
- Edit Solution: Users can edit the fields of an unsupervised/supervised solution they created using the web app's graphical user interface or API endpoints.
- Delete Solution: Users can delete an unsupervised/supervised solution they created using the web app's graphical user interface or API endpoints.
- Edit Scenario: An admin user can edit the fields of a scenario belonging to a specific user by providing their username and scenario name via the web app's graphical user interface or API endpoints.
- Delete Scenario: An admin user can delete a scenario belonging to a specific user by providing their username and scenario name via the web app's graphical user interface or API endpoints.
- View All Solutions: An admin user can view all supervised/unsupervised solutions belonging to a specific user by providing their username via the web app's graphical user interface or API endpoints.
- Get Specific Solution: An admin user can get a specific supervised/unsupervised solution belonging to a specific user by providing their username and solution name via the web app's graphical user interface or API endpoints.
- Get Specific Scenario: An admin user can get a specific scenario belonging to a specific user by providing their username and scenario name via the web app's graphical user interface or API endpoints.

3.5 Web technologies

3.5.1 Frontend

Several frontend technologies could be suitable for a trusted AI platform. Following are a few options to consider and some of their key features.

The comparison of React, Angular, and Vue, along with various criteria like popularity, performance, learning curve, community support, Syntax, architecture, Full-featured framework, and mobile development, is presented in Table3.1.

Table 3.1: Comparison between React, Angular, and Vue

Criteria	React	Angular	Vue
Popularity	High	Moderate	High
Performance	Good	Good	Good
Learning curve	Moderate	Steep	Easy
Community support	Strong	Strong	Strong
Syntax	JSX	HTML/TypeScript	HTML/JavaScript
Architecture	Component-based	Component-based	Component-based
Full-featured framework	No	Yes	No
Mobile development	Yes	Yes	Yes

React

React is a JavaScript library for building user interfaces. It is lightweight, easy to learn, and has a large developer community. React's virtual DOM (Document Object Model) allows for efficient components rendering, and its functional programming style makes it easy to reason about code.[18]

Angular

Angular is a comprehensive framework for building single-page applications. It uses a declarative template syntax, dependency injection, and a reactive programming style to make it easy to build complex applications. Angular has a large developer community and many built-in features, such as a router and a form validation library.[10]

Vue

Vue is a progressive JavaScript framework that is easy to learn and use. It uses a template syntax similar to HTML, and its reactive components make it easy to build interactive user interfaces. Vue.js has a lightweight run-time designed to integrate easily into existing projects.[15]

3.5.2 Backend

Several backend technologies could be suitable for a trusted AI platform. Following are a few options to consider and some of their key features.

Criteria	Django	Laravel	Spring
Programming language	Python	PHP	Java
Popularity	High	High	High
Learning curve	Steep	Steep	Steep
Community support	Strong	Strong	Strong
Syntax	Python	PHP	Java
Database support	Wide range	Wide range	Wide range
Scalability	Good	Good	Good
Security features	Strong	Strong	Strong

Django

Django is a high-level web development framework written in Python. It is known for its strong focus on security and its wide range of database support. Django is designed to be easy to use and scalable, and it has a large and active community of developers. It uses a model-template-view (MTV) architecture and is well-suited for building complex, data-driven web applications.[8]

Laravel

Laravel is a PHP web development framework known for its elegant syntax and wide range of database support. It is designed to be easy to use and scalable, and it has a large and active community of developers. Laravel uses a model-view-controller (MVC) architecture and is well-suited for building various web applications, including e-commerce platforms and content management systems.[16]

Spring

Spring is a Java-based web development framework known for its strong scalability and security features. It is designed to be flexible and modular, and it has a large and active community of developers. Spring uses an inversion of control (IoC) container and is well-suited to build various web applications, including enterprise-level and microservices.[9]

Chapter 4

Design

The web application will be built using the Angular frontend framework and the Django backend framework. Angular provides a powerful and modular front-end architecture that makes it easy to create reusable components, while Django provides a well-structured backend architecture with built-in authentication and database support. The high-level architecture of the application will consist of three main components: the frontend, the backend API, and the database.

The frontend of the application will be built using the Angular framework. Angular is a popular front-end framework that allows developers to create reusable UI components using a combination of HTML, CSS, and TypeScript. Angular's component-based architecture makes it easy to create modular and reusable UI components that can be used throughout the application. The frontend of the application will be responsible for presenting the user interface to the user, handling user interactions, and making API requests to the backend.

The backend of the application will be built using the Django framework. Django is a popular backend framework that provides built-in support for authentication, user management, database models, and API development. The backend of the application will be responsible for handling API requests from the frontend, performing business logic, and interacting with the database. The backend will be designed using a RESTful API architecture, which will allow clients to interact with the application using HTTP requests.

The application will use a MySql (Sqlite) database to store user data, scenario data, and solution data. Sqlite is a lightweight and reliable database system that is widely used in web development. MySQL is a popular open-source relational database management system (RDBMS) that uses Structured Query Language (SQL) to manage and manipulate data. The database will be accessed using Django's ORM (Object-Relational Mapping) system, which provides a high-level interface for interacting with the database[20].

The user management system will be implemented using Django's built-in authentication system. Admin users will have access to a Django admin site, which will allow them to manage users, scenarios, and solutions. Non-admin users will be able to register for an account, log in, and manage their own scenarios and solutions.

4.1 Scenarios & Solutions

Scenarios will be stored in the database as a table of Scenario objects, which will have fields for the scenario name, description, and owner (user ID). Users will be able to create new scenarios by submitting a form with the scenario name and description, which will be sent to the backend API. The backend API will create a new Scenario object and associate it with the current user. Users will be able to view their own scenarios on a dashboard page, and admin users will be able to view all scenarios created by a specific user.

Solutions will be stored in the database as a table of Solution objects, which will have fields for the solution name, solution type, training file, test file, and other solution details. Users will be able to create new solutions by submitting a form with the solution details, which will be sent to the backend API. The backend API will create a new Solution object and associate it with the current user and the corresponding scenario. Users will be able to view their own solutions on a dashboard page, and admin users will be able to view all solutions created by a specific user.

4.2 API Endpoints

The backend API will expose several API endpoints for interacting with the application data. These endpoints will be designed using a RESTful API architecture, which will allow clients to interact with the application using HTTP requests. Examples of API endpoints include:

- GET /scenario - retrieve a list of all scenarios
- GET /scenario/scenario_name - get a a scenario
- POST/scenario - create a new scenario
- DEL/scenario/scenario_name - delete a a scenario
- PUT/scenario/scenario_name - update a a scenario
- GET /solution - retrieve a list of all solutions
- GET /solution/solution_name - get a a solution
- POST /solution - create a new solution
- DEL /solution/solution_name - delete a a solution
- PUT /solution/solution_name - update a a solution
- POST /metric_name_score/solution_name - get the metric score for a solution
- POST /pillar_name_score/solution_name - get the metric score for a solution

These API endpoints will be secured using token-based authentication, which will require clients to login with their credentials in in each API request.

4.3 UI Components

The UI components of the application will be built using Angular's component-based architecture. Examples of UI components include:

- Login and registration forms
- Scenario creation form
- Scenario list component
- Solution creation form
- Solution list component
- Analyze solutions page
- Compare solutions page
- Dashboard page for viewing user scenarios and solutions
- Django admin pages for managing users, scenarios, and solutions

These UI components will be designed using responsive web design principles, which will ensure that the application is accessible and usable on a variety of devices and screen sizes.

4.4 Authorization & Error Handling

The backend API will extend djangos BaseAuthentication subclass to secure access to the application data. The authenticate method is responsible for authenticating the user and retrieves a HTTP Authorization header from the META request data and decodes it using Base64 splitting it up in username and password. If the username is not empty it will check for the existence of the user and check the password with the django_pbkdf2_sha256 library, using a secure password hashing algorithm.

The frontend and backend will implement a consistent error-handling strategy. When an error occurs, the frontend will display a user-friendly error message, and the backend will log the error and return an appropriate HTTP status code. The frontend and backend will communicate errors using a standardized error response format, which will include an error code and a human-readable error message.

4.5 Figma Frames

Dashboard with Sidebar: This frame shows the dashboard for the web application, which features a sidebar with links to different pages. The main section of the dashboard provides an overview of different metrics, including scenarios, solutions, and the last calculated trust metric score.

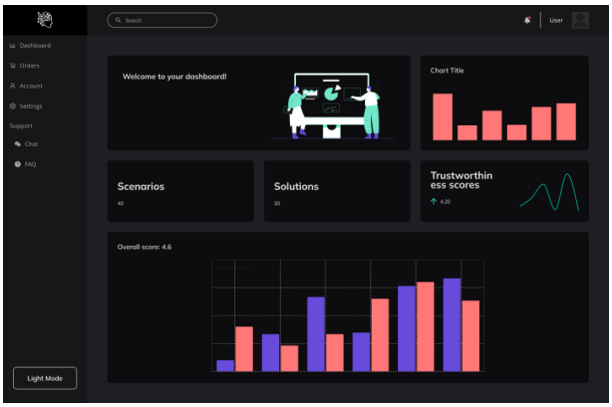

Figure 4.1: Dashboard with Sidebar

Hamburger Menu

This frame displays the hamburger menu, which can be used to navigate to different pages of the web application. The menu provides links to Scenarios, Upload, Analyze, Compare, and Users pages.

Figure 4.2: Hamburger Menu

Login Page

This frame displays the login page for the web application, where users can enter their username and password to access the site's features.

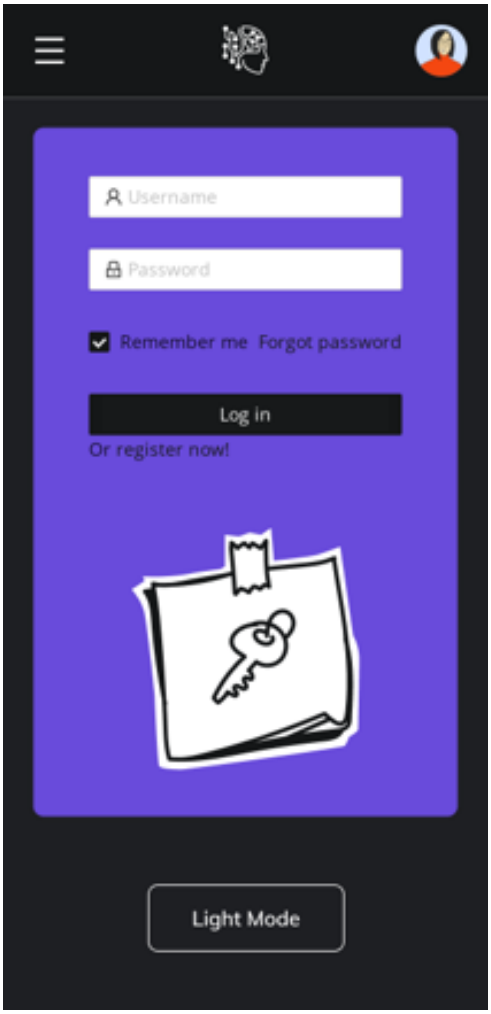

Figure 4.3: Login Page

Mobile View of Dashboard

This frame shows a mobile view of the dashboard, featuring a graphical representation of the trust score and pillar scores.

Figure 4.4: Mobile View of Dashboard

4.6 System Architecture

The backend of the web application is built using the Django web framework, which follows a Model-View-Controller (MVC) architectural pattern. In this pattern, the Model represents the data and database schema, the View represents the user interface, and the Controller handles user input and updates the Model and View accordingly. Django uses a similar pattern called Model-View-Template (MVT), where the Template represents the View.

The application's backend is structured using a modular design, where each module handles a specific function of the application. The main modules in the backend include:

- **Models:** This module defines the data models used by the application and specifies the database schema. The models are defined in the `models` folder, and Django automatically generates the necessary database tables based on the model definitions. The data models used in the application include `CustomUser`, `Scenario`, and `ScenarioSolution`.

- **Authentication:** This module provides authentication functionality for the application. The authentication logic is defined in the `authentication.py` file. The application uses token-based authentication to secure the API endpoints.

- **Admin:** This module provides an administrative interface for managing the application. The administrative interface is defined in the `admin.py` file. The `CustomUser`, `Scenario`, and `ScenarioSolution` models are registered with the Django admin interface, allowing the administrator to view, add, edit, and delete instances of these models.

- **Serializers:** This module provides serialization and deserialization functionality for the application. The serialization logic is defined in the `serializers.py` file. The serializers are used to convert complex data types, such as database objects, into JSON format for use in the API endpoints.
- **Apps** This module is used to configure the `ApisConfig` class, which defines the configuration settings for the `apis` app. The configuration settings include the default auto field used for database model primary keys.
- **Views:** This module handles user requests and generates responses. The views are defined in the `views.py` file and use Django's template system to render HTML pages.
- **URLs:** This module maps URLs to view functions. The URLs are defined in the `urls.py` file and use Django's URL routing system to handle requests.

The frontend of the web application is built using Angular, a popular web framework for building single-page applications. Angular uses a component-based architecture, where each component represents a reusable piece of UI functionality. The main modules in the frontend include:

- **Components:** This module defines the UI components used by the application. The components are implemented in TypeScript files and HTML templates, and can be customized with CSS styles.
- **Services:** This module provides shared functionality across components, such as data retrieval and manipulation. The services are implemented in TypeScript files and can be injected into components to provide data and functionality [10].

4.7 Performance Design

Performance is a crucial aspect of any web application. In Django, several built-in features can be used to design a performant backend. One important module is Django's caching framework. The caching framework can help reduce the number of database queries, which can speed up response times.

```
# settings.py

CACHES = {
    'default': {
        'BACKEND': 'django.core.cache.backends.db.DatabaseCache',
        'LOCATION': 'cache_table',
        'TIMEOUT': 3600, # in seconds
        'OPTIONS': {
            'MAX_ENTRIES': 10000
        }
    }
}
```

This example uses the DatabaseCache backend, which stores cached data in a database table. The LOCATION parameter specifies the name of the table to use for caching. The TIMEOUT parameter specifies how long the cached data should be stored (in seconds), and the 'MAX_ENTRIES' parameter specifies the maximum number of entries that can be stored in the cache[11].

Another important module in Django for performance is the middleware system. Custom middleware can be added to the middleware stack to perform various tasks, such as compressing responses, caching API responses, or throttling requests from specific IP addresses.

On the Angular frontend, performance can be improved by using several built-in techniques. One important technique is lazy loading of modules. Lazy loading allows Angular to load modules on demand, which can reduce the initial load time of the application[10].

```
// app-routing.module.ts

const routes: Routes = [
  { path: '', redirectTo: 'home', pathMatch: 'full' },
  { path: 'home', loadChildren: () => import('./home/home.module').then(m => m.
      HomeModule) },
  { path: 'about', loadChildren: () => import('./about/about.module').then(m => m
      .AboutModule) },
  { path: 'contact', loadChildren: () => import('./contact/contact.module').then(
      m => m.ContactModule) },
];
```

In this example, the loadChildren property is used to lazy load modules. The import function is used to dynamically load the module when the user navigates to the corresponding route.

Another technique for improving performance in Angular is preloading of data. Preloading can be used to load data in the background before the user requests it, which can improve perceived performance[10].

```
// app-routing.module.ts

const routes: Routes = [
  { path: '', redirectTo: 'home', pathMatch: 'full' },
  { path: 'home', component: HomeComponent, data: { preload: true } },
  { path: 'about', component: AboutComponent },
  { path: 'contact',

\begin{lstlisting}[language=Python]
```

4.7.1 Security Design

1. Password Hashing in Django Django automatically hashes user passwords before storing them in the database, which helps protect user accounts in case the database is compromised. To hash a password in Django, you can use the make$_p asswordf unctionf romthe$

```
from django.contrib.auth.hashers import make_password

password = 'my_password'
hashed_password = make_password(password)
\\end{lstlisting}

\item HTTPS Support in Django
Django includes built-in support for HTTPS, which helps protect data in
    transit. To use HTTPS in Django, you can simply enable HTTPS on your
    web server and set the SECURE_SSL_REDIRECT setting to True in your
    Django settings:
\begin{lstlisting}[language=Python]

SECURE_SSL_REDIRECT = True
\end{verbatim}
```

4.8 Usability Design

1. Built-in Administration Interface in Django Django provides a built-in administration interface, which allows developers to manage application data without having to write custom views or templates. To use the Django administration interface, you simply need to register your models with the admin site:

```
from django.contrib import admin
from .models import MyModel

admin.site.register(MyModel)
```

2. Responsive Design in Angular Angular provides built-in support for responsive design, which allows developers to create applications that can adapt to different screen sizes and devices. To use responsive design in Angular, you can use CSS media queries and Angular's @media rule to apply styles based on the device screen size:

```
@media only screen and (max-width: 600px) {
  /* styles for mobile devices */
}

@media only screen and (min-width: 600px) {
  /* styles for desktop devices */
}

<div [ngClass]="{'mobile': isMobile(), 'desktop': isDesktop()}">
  <!-- content here -->
</div>
```

3. Angular Material Design Angular provides built-in support for Material Design, a design language developed by Google. Angular Material includes a set of pre-built UI components, such as buttons, cards, and menus, that can be easily added to an application. To use Angular Material in an application, one can install the @angular/material package and import the components.

4. Form Validation in Angular Angular provides built-in form validation, which allows developers to validate user input and provide feedback to the user. To use form validation in Angular, you can use built-in validators, such as required and email, or create custom validators:

```
<form #myForm="ngForm">
  <input type="text" name="email" ngModel required email>
  <div *ngIf="myForm.controls.email.invalid␣&&␣(myForm.controls.email.
      dirty␣||␣myForm.controls.email.touched)">
    <div *ngIf="myForm.controls.email.errors.required">Email is required
        .</div>
    <div *ngIf="myForm.controls.email.errors.email">Email is invalid.</div
        >
```

```
  </div>
</form>
```

Chapter 5

Implementation

The scenario, supervised solution models are based on the definitions created in the master project by Joel Leupp, Melike Demirci and Jan Bauer, the unsupervised solution model is based on the definition implemented by Mauro Doerig in his bachelor thesis. The Trust-,Pillar-, Metricscore functions are (somehow) related to those created in the last iteration of the Trusted-AI webapp. The error prone loading of joblib model files due to corrupt dumping of trained models or mismatches in python version lead to many changes.

5.1 Scenarios & Solutions

The `scenario` class inherits from the `APIView` class of the Django REST framework. It defines three methods for handling HTTP requests: `get`, `put`, and `post`.

```
class scenario(APIView):
def get(self, request, scenarioId):
    scenario = Scenario.objects.get(id=scenarioId)

    print('id:', scenario.description, scenario.scenario_name)
    if (scenario is not None):
        return Response({
            'scenarioName': scenario.scenario_name,
            'description': scenario.description,
        }, status=200)
    else:
        return Response("Not␣Exist", status=201)

def put(self, request):
    scenario = Scenario.objects.get(
        id=request.data['id'])

    scenario.scenario_name = request.data['name']
    scenario.description = request.data['description']
    scenario.save()
```

```
    return Response("successfully changed")

def post(self, request):

    user = CustomUser.objects.get(email=request.data['emailid'])
    try:
        newScenario = Scenario.objects.create(
            scenario_name=request.data['ScenarioName'],
            description=request.data['Description'],
            user_id=user.id,
        )

        newScenario.save()

        return Response({'Save Success'}, status=200)
    except:
        return Response({'Save Failed'}, status=400)
```

The `get` method retrieves a scenario instance from the database based on its ID, which is passed as a URL parameter. If the scenario exists, the method returns a JSON response with the scenario's name and description, and a status code of 200. Otherwise, it returns a response with a message of "Not Exist" and a status code of 201.

The `put` method updates an existing scenario instance with new data passed in the request body. The method retrieves the scenario based on its ID, and then updates its name and description attributes with the values from the request body. Finally, the updated scenario is saved to the database, and a response with a message of "successfully changed" is returned.

The `post` method creates a new scenario instance in the database based on data passed in the request body. The method retrieves the user associated with the email passed in the request body, and then creates a new scenario instance with the scenario name, description, and user ID from the request body. Finally, the new scenario is saved to the database, and a response with a message of "Save Success" is returned. If there is an error creating the scenario instance, a response with a message of "Save Failed" and a status code of 400 is returned.

These API views fulfill the requirements of the use cases for interacting with the Scenario model. The `get` method allows users to retrieve a scenario by its ID, which is useful for displaying information about a specific scenario. The `put` method allows users to update an existing scenario, which is necessary for making changes to a scenario. The `post` method allows users to create a new scenario, which is necessary for adding new scenarios to the system. The use of Django REST framework's built-in `APIView` class and the `Response` class simplifies the implementation of the API views, making the code more readable and maintainable.

The solution class is responsible for handling the CRUD (Create, Read, Update, Delete) operations related to the solutions of the scenarios. The GET method of the solution

class retrieves the names of the solutions uploaded by a user using their email address. The POST method is responsible for adding a new solution to the database.

```
class solution(APIView):
    def get(self, request, email):
        uploaddic = {}

        SolutionName = []

        userexist = CustomUser.objects.filter(email=email)
        if userexist:
            userobj = CustomUser.objects.get(email=email)
            scenarioobj = ScenarioSolution.objects.filter(
                user_id=userobj.id).values()

            if scenarioobj:
                for i in scenarioobj:
                    SolutionName.append(i['solution_name'])

            uploaddic['SolutionName'] = SolutionName
            return Response(uploaddic)

        else:
            return Response("User␣not␣exist....␣Please␣Sign␣Up!")

    def post(self, request):
        if request.data is not None:
            mapFile = ''
            if request.data['MapFile'] is None or request.data['MapFile'] == '
                undefined':
                mapFile = 'files/mapping_metrics_default.json'
            else:
                mapFile = request.data['MapFile']

            print('req␣dta:', request.data)

            try:
                userexist = CustomUser.objects.get(
                    email=request.data['emailid'])
                scenario = Scenario.objects.get(
                    scenario_name=request.data['SelectScenario'])
                fileupload = ScenarioSolution.objects.create(
                    user_id=userexist.id,
                    scenario_id=scenario.id,
                    solution_name=request.data['NameSolution'],
                    description=request.data['DescriptionSolution'],
                    training_file=request.data['TrainingFile'],
                    metrics_mappings_file=mapFile,
                    test_file=request.data['TestFile'],
                    factsheet_file=request.data['FactsheetFile'],
```

```
                model_file=request.data['ModelFile'],
                target_column=request.data['Targetcolumn'],
                solution_type=request.data['Solutiontype'],

                outlier_data_file=request.data['Outlierdatafile'],
                protected_features=request.data['ProtectedFeature'],
                protected_values=request.data['Protectedvalues'],
                favourable_outcome=request.data['Favourableoutcome'],
                weights_metrics=request.data['WeightMetric'],
                weights_pillars=request.data['WeightPillar']
            )
            fileupload.save()

            return Response("Successfully␣add!", status=200)

        except Exception as e:
            print('errror:', e)
            return Response("Error␣occured", status=201)
```

For the GET method, the implementation starts by initializing an empty dictionary called uploaddic and an empty list called SolutionName. The code then checks if the user exists in the database by filtering the CustomUser model using their email. If the user exists, the code retrieves their object from the database and then queries the ScenarioSolution model to get all the solutions uploaded by the user using their user ID. If solutions are found, the names of the solutions are appended to the SolutionName list. Finally, the SolutionName list is added to the uploaddic dictionary using the key 'SolutionName'. The method returns a response containing the uploaddic dictionary.

For the POST method, the implementation starts by checking if the request data is not None. If the data is not None, the code checks if the MapFile key is present in the request data. If it is not present or its value is 'undefined', the mapFile variable is set to $'files/mapping_metrics_default.json'$, otherwise, it is set to the value of the MapFile key. The code then tries to retrieve the user object from the database using their email address and the scenario object using the scenario name provided in the request data. After that, a new object is created in the ScenarioSolution model using the data provided in the request. The method returns a response with the message 'Successfully add!' if the solution is added successfully, otherwise it returns a response with the message 'Error occurred'.

To implement the solution class, Django's APIView module is used, which allows us to define the HTTP methods for a particular endpoint. The module also provides the Response class, which is used to send a response to the client. The CustomUser, Scenario, and ScenarioSolution models are used to store the user, scenario, and solution data in the database. The filter and get methods are used to retrieve the user object from the CustomUser model and the scenario object from the Scenario model respectively. The create method is used to create a new object in the ScenarioSolution model.

Overall, the solution class fulfills the requirements of allowing users to upload their solutions to a particular scenario and retrieve the names of the solutions uploaded by them.

The implementation follows the best practices of the Django framework and provides an efficient and secure way of storing and retrieving data from the database.

5.2 Pillar Functions

The implementation of functions that return the scores for the four Pillars of Trust Explainability, Fairness, Accountability / Methodology, Robustness (Explainbilityscore, Fairnesscore, Accountabilityscore, Robustnesscore) for unsupervised or supervised solutions are based in the algorithms (un)supervised Functions Pillarname folder and there saved as Pillarname.py. There parameters are displayed here:

The pillar functions take following inputs:

- model: trained model object
- training_dataset: training dataset file path
- test_dataset: test dataset file path
- factsheet: factsheet file path
- mappings: mappings file path
- target_column: target column name
- outliers_data: outliers data file path
- thresholds: dictionary consisting of threshold values
- outlier_thresholds: dictionary consisting of outlier threshold values
- outlier_percentage: outlier percentage value
- high_cor: high correlation threshold value
- print_details: boolean value indicating whether to print the details of each score or not

The pillar functions use the input parameters to compute the pillar scores by calling the corresponding functions imported from the other Python modules. They return as a result object consisting of score and properties for each of the metric scores they are composed of. The score is a float value indicating the score obtained for that particular accountability score. The properties are a dictionary consisting of additional information related to that score.

5.2.1 Accountability Score

The `get_accountability_score_supervised` function imports various helper functions from the accountability folder and calculates the following scores:

- normalization score: measures the degree of normalization of the data
- missing data score: measures the degree of missing data in the dataset
- regularization score: measures the degree of regularization applied to the model
- train test split score: measures the degree of randomness in the train-test split
- factsheet completeness score: measures the completeness of the factsheet

The `get_accountability_score_unsupervised` function computes the same scores as the function `get_accountability_score_supervised`.

The function returns a result object consisting of score and properties for each of the above scores. The score is a float value indicating the score obtained for that particular accountability score. The properties are a dictionary consisting of additional information related to that score.

5.2.2 Explainability Score

The `get_explainability_score_supervised` function imports various helper functions from the explainability folder and calculates the following scores:

- algorithm class score: evaluates the model's explainability degree based on algorithm type and complexity
- correlated features score: measures the percentage of highly correlated features.
- model size score: calculates the number of parameters used by models.
- feature relevance score: computes the percentage of irrelevant features for a set of predictions.

The `get_accountability_score_unsupervised` function computes the correlated features model size score as well as the permutation feature importance score, measuring the importance of features by iterating over each feature column.

5.2.3 Fairness Score

The `get_fairness_score_supervised` function imports various helper functions from the fairness folder and calculates the following scores:

- underfitting: calculates the difference between train and baseline performance.
- overfitting: computes the difference between train and test performance, giving the model.
- class balance: measures the ratio of samples belonging to different classes in the trainin dataset. parity difference: computes the spread between the percentage of samples receiving a favorable outcome for protected and unprotected samples groups.
- equal opportunity difference: measures the spread between true positive rate (TPR) and false positive rate (FPR) between different groups.

The `get_fairness_score_unsupervised` function computes only the underfitting, overfitting, statistical parity difference and disparate impact score.

5.2.4 Robustness Score

The `get_robustness_score_supervised` function imports various helper functions from the robustness folder and calculates the following scores:

- CLEVER Score: estimates the minimal perturbation required to change the classification for a given input. The algorithm uses an estimate based on extreme value theory.
- Confidence Score: measures the probability of correctly predicting a given sample.
- Clique Method: finds the exact minimal perturbation required to change a classification outcome.
- ER Fast Gradient: a white-box attack that is effective on Logistic Regression models, Support Vector Classifiers, and Neural Networks.
- ER Carlini Wagner: a white-box targeted attack algorithm tailored to three distance metrics. It optimizes a minimization problem using gradient descent.
- ER DeepFool: an efficient white-box and untargeted attack for deep neural networks. It finds the nearest decision boundary in norm for a given input and can be modified to be effective on Logistic Regression models and Support Vector Classifiers.
- Loss Sensitivity: local loss sensitivity quantifies the smoothness of a model by estimating its Lipschitz continuity constant. The smaller the value, the smoother the function.

The `get_robustness_score_unsupervised` function computes only the CLEVER score.

5.3 Metric Functions

The metric functions are closely related to the work of the previous two student(s).

5.4 APIs

The API endpoints for the accountability, fairness, explainability and robustness metrics take as parameters the solution_name. Post requests return the metric scores of the same name. The handle_request function calls depending on the api_endpoint and the solution_type of the solution of given solution_name the metric function that has to be called. The pillars and trustscore API Endpoints are implemented in the same manner.

5.4.1 APIs Accountability

```
@ parser_classes([MultiPartParser, FormParser])
@ api_view(['POST'])
@ authentication_classes([CustomUserAuthentication])
def get_factsheet_completness_score(request):
    return handle_score_request('account', 'factsheet', request.data, request.
        user.id)


@ parser_classes([MultiPartParser, FormParser])
@ api_view(['POST'])
@ authentication_classes([CustomUserAuthentication])
def get_missing_data_score(request):
    return handle_score_request('account', 'missingdata', request.data, request.
        user.id)


@ parser_classes([MultiPartParser, FormParser])
@ api_view(['POST'])
@ authentication_classes([CustomUserAuthentication])
def get_normalization_score(request):
    return handle_score_request('account', 'normalization', request.data, request
        .user.id)


@ parser_classes([MultiPartParser, FormParser])
@ api_view(['POST'])
@ authentication_classes([CustomUserAuthentication])
def get_regularization_score(request):
    return handle_score_request('account', 'regularization', request.data,
        request.user.id)
```

```
@ parser_classes([MultiPartParser, FormParser])
@ api_view(['POST'])
@ authentication_classes([CustomUserAuthentication])
def get_train_test_split_score(request):
    return handle_score_request('account', 'train_test', request.data, request.
        user.id)
```

5.4.2 APIs Explainability

```
# 1)ModelSizeScore
@ parser_classes([MultiPartParser, FormParser])
@ api_view(['POST'])
@ authentication_classes([CustomUserAuthentication])
def get_modelsize_score(request):
    return handle_score_request('explain', 'modelsize_score', request.data,
        request.user.id)

# 2)CorrelatedFeaturesScore


@ parser_classes([MultiPartParser, FormParser])
@ api_view(['POST'])
@ authentication_classes([CustomUserAuthentication])
def get_correlated_features_score(request):
    return handle_score_request('explain', 'correlated_features_score', request.
        data, request.user.id)


# 3)AlgorithmClassScore

@ parser_classes([MultiPartParser, FormParser])
@ api_view(['POST'])
@ authentication_classes([CustomUserAuthentication])
def get_algorithm_class_score(request):
    return handle_score_request('explain', 'algorithm_class_score', request.data,
         request.user.id)


# 4)FeatureRelevanceScore

@ parser_classes([MultiPartParser, FormParser])
@ api_view(['POST'])
@ authentication_classes([CustomUserAuthentication])
def get_feature_relevance_score(request):
    return handle_score_request('explain', 'feature_relevance_score', request.
        data, request.user.id)
```

```
# 5)PermutationFeatures

@ parser_classes([MultiPartParser, FormParser])
@ api_view(['POST'])
@ authentication_classes([CustomUserAuthentication])
def get_permutation_feature_importance_score(request):
    return handle_score_request('explain', 'permutation_feature_importance_score'
        , request.data, request.user.id)
```

5.4.3 APIs Fairness

```
# D)Fairness
# 1)DisparateImpactScore
@ parser_classes([MultiPartParser, FormParser])
@ api_view(['POST'])
@ authentication_classes([CustomUserAuthentication])
def get_disparate_impact_score(request):
    return handle_score_request('fairness', 'disparate_impact_score', request.
        data, request.user.id)


# 2)ClassBalanceScore
@ parser_classes([MultiPartParser, FormParser])
@ api_view(['POST'])
@ authentication_classes([CustomUserAuthentication])
def get_class_balance_score(request):
    return handle_score_request('fairness', 'disparate_impact_score', request.
        data, request.user.id)


# 3)OverfittingScore

@ parser_classes([MultiPartParser, FormParser])
@ api_view(['POST'])
@ authentication_classes([CustomUserAuthentication])
def get_overfitting_score(request):
    return handle_score_request('fairness', 'overfitting_score', request.data,
        request.user.id)


# 4)UnderfittingScore

@ parser_classes([MultiPartParser, FormParser])
@ api_view(['POST'])
@ authentication_classes([CustomUserAuthentication])
def get_underfitting_score(request):
    return handle_score_request('fairness', 'underfitting_score', request.data,
        request.user.id)
```

```
# 5)StatisticalParityDifferenceScore
@ parser_classes([MultiPartParser, FormParser])
@ api_view(['POST'])
@ authentication_classes([CustomUserAuthentication])
def get_statistical_parity_difference_score(request):
    return handle_score_request('fairness', 'statistical_parity_difference_score'
        , request.data, request.user.id)


# 6)EqualOpportunityDifferenceScore
@ parser_classes([MultiPartParser, FormParser])
@ api_view(['POST'])
@ authentication_classes([CustomUserAuthentication])
def get_equal_opportunity_difference_score(request):
    return handle_score_request('fairness', 'equal_opportunity_difference_score',
         request.data, request.user.id)


# 7)AverageOddsDifferenceScore
@ parser_classes([MultiPartParser, FormParser])
@ api_view(['POST'])
@ authentication_classes([CustomUserAuthentication])
def get_average_odds_difference_score(request):
    return handle_score_request('fairness', 'average_odds_difference_score',
        request.data, request.user.id)
```

5.5 Pages

5.5.1 Dashboard & Analyze & Compare Pages

Following the Analyze page backend is described, the dashboard and compare page have a similar structure. The Analyze page inherits from the APIView class of the Django REST framework. This class has two methods: get() and post().

The get() method takes two arguments, request and id, and prints a message "User not exist.... Created new". It doesn't return anything.

The post() method takes one argument, request, and initializes an empty dictionary called uploaddic. It then retrieves the user from the CustomUser model with a username specified in the userid field of the request data. It also retrieves the scenario from the Scenario model with a scenario name specified in the SelectScenario field of the request data, and gets a ScenarioSolution object with the user_id matching the retrieved user's id. It then populates the ScenarioName and Description lists with the scenario name and description, respectively, if the scenario name matches the one specified in the request data.

After that, the method defines a function called get_performance_metrics_supervised() which takes five arguments: model, test_data, target_column, train_data, and factsheet. This function first tries to read the model file using pd.read_pickle(). If this fails, it prints a message "MODEL ERROR". It then reads the test_data and train_data files using pd.read_csv(), and reads the factsheet file using pandas.read_json(). It then loads the factsheet file using the json.loads() function.

The function then calculates various performance metrics such as accuracy, recall, precision, and F1 score using the metrics module of the scikit-learn library. It stores these metrics in the uploaddic dictionary.

The get_performance_metrics_unsupervised function calculates various performance metrics sfor unsupervised solutions.

The post() method checks if the factsheet dictionary contains a key called "properties". If it does, it extracts some properties from the factsheet and stores them in the properties dataframe. It also stores some properties from the factsheet in the uploaddic dictionary.

Overall, the post() method prints the results on the analyze page.

The function get_final_score() calculates the final score of the model

First, the function loads the configurations from the mappingConfig file which contains information on how to analyze each metric. Then it calls the trusting_AI_scores() function which returns the scores and properties of the model for each metric.

If the default_map is called, the function checks whether the scores and properties are already in the factsheet file, and if so, it uses them directly. Otherwise, it calls the trusting_AI_scores() function to compute the scores and properties and save them to the factsheet file.

After computing the scores for each metric, the function calculates the final score using weights from the config_weights file. For each metric, the function extracts the relevant scores and properties from the output of the trusting_I_scores() function and calculates the weighted score. The final scores are stored in the final_scores dictionary, which is returned by the function.

5.5.2 Scenario & Solution Detailspage

The scenariodetail page is identical to the scenario creation page.

The solutiondetail class is an implementation of an API view that handles HTTP GET and PUT requests for a solution detail page.

The get method retrieves a single ScenarioSolution object based on the id provided in the request URL and returns a JSON response with various fields from that object.

The delete method deletes a single ScenarioSolution object based on the id provided in the request URL and returns a JSON response indicating success.

The put method updates the solution details in the database based on the solution ID passed in the request data. It retrieves the existing solution details from the database and updates the attributes based on the request data. If any of the attributes are not present in the request data, they are not updated. Once the attributes are updated, the changes are saved to the database and a success message is returned with a status code of 200.

The get method retrieves the solution details from the database based on the id parameter passed in the request URL. It then returns a JSON response that includes several attributes of the ScenarioSolution object such as solution name, description, solution type, protected features, protected values, and target column. The status code of the response is 200.

```
class solutiondetail(APIView):
    def get(self, request, id):
        solutionDetail = ScenarioSolution.objects.get(id=id)

        return Response({
            'solution_name': solutionDetail.solution_name,
            'description': solutionDetail.description,
            'solution_type': solutionDetail.solution_type,
            'protected_features': solutionDetail.protected_features,
            'protected_values': solutionDetail.protected_values,
            'target_column': solutionDetail.target_column
        }, status=200)

    def delete(self, request, id):
        print('delete␣id:', id)
        solutiondetail = ScenarioSolution.objects.get(id=id).delete()

        return Response({
            'Delete␣ok',
        }, status=200)

    def put(self, request):
        solutionDetail = ScenarioSolution.objects.get(
            id=request.data['SolutionId'])
        solutionDetail.solution_name = request.data['NameSolution']
```

```
solutionDetail.description = request.data['DescriptionSolution']
if (request.data['TrainingFile'] != 'undefined'):
    print('asdfasdfasdf')
if (request.data['TrainingFile'] != 'undefined'):
    solutionDetail.training_file = request.data['TrainingFile']
if (request.data['TestFile'] != 'undefined'):
    solutionDetail.test_file = request.data['TestFile']
if (request.data['FactsheetFile'] != 'undefined'):
    solutionDetail.factsheet_file = request.data['FactsheetFile']
if (request.data['ModelFile'] != 'undefined'):
    solutionDetail.model_file = request.data['ModelFile']
if (len(request.data['Targetcolumn']) <= 0):
    solutionDetail.target_column = request.data['Targetcolumn']
if (request.data['Outlierdatafile'] != 'undefined'):
    solutionDetail.outlier_data_file = request.data['Outlierdatafile']
if (len(request.data['ProtectedFeature']) <= 0):
    solutionDetail.protected_features = request.data['ProtectedFeature']
if (len(request.data['Protectedvalues']) <= 0):
    solutionDetail.protected_values = request.data['Protectedvalues']
if (len(request.data['Favourableoutcome']) <= 0):
    solutionDetail.favourable_outcome = request.data['Favourableoutcome']
if (len(request.data['WeightMetric']) <= 0):
    solutionDetail.weights_metrics = request.data['WeightMetric']
if (len(request.data['WeightPillar']) <= 0):
    solutionDetail.weights_pillars = request.data['WeightPillar']
solutionDetail.save()

return Response('successfully␣changed', 200)
```

5.6 Graphical Userinterface

When acessing the website under ritual-ai.net the user is redirected to the singup/login page:

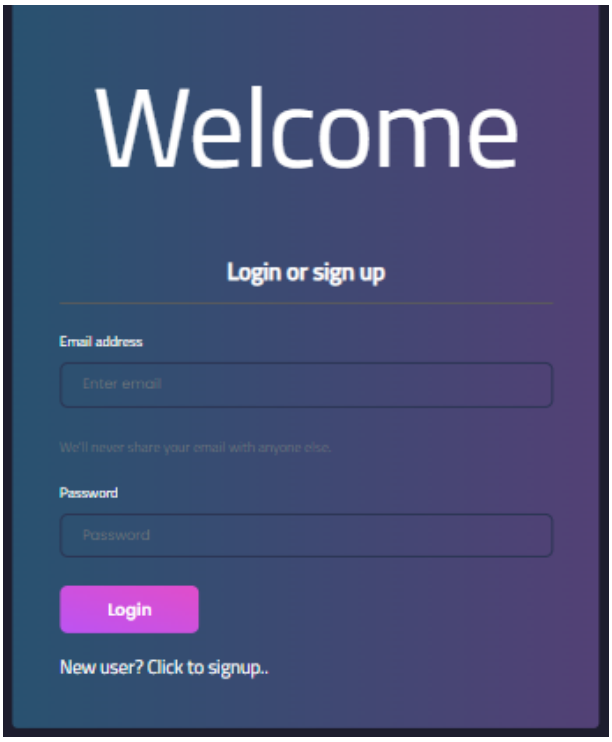

Figure 5.1: Login

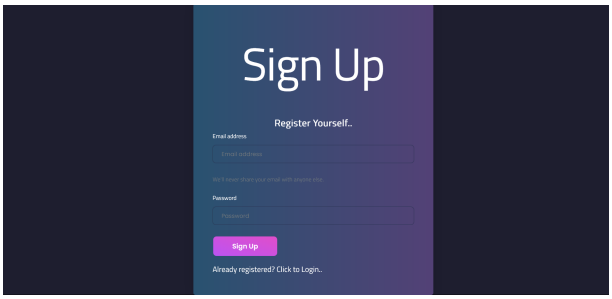

Figure 5.2: Signup

After singup with a username and password and login with those the user is redirected to the dashboard. In the sidebar the user can by click on one of the sidebar items acesss the scenario creation page After filling out scenario name, description the user is redirected to the the scenario upload page where he can choose by click on the unsupervised/supervised button if he wants to upload a supervised or unsupervised solution, a click displays the form corresponding to the model of those solution types.

After clicking upload solution the user is redirected to the dashboard where after the scores are calculated they are displayed, for supervised solutions in the supervised solutions

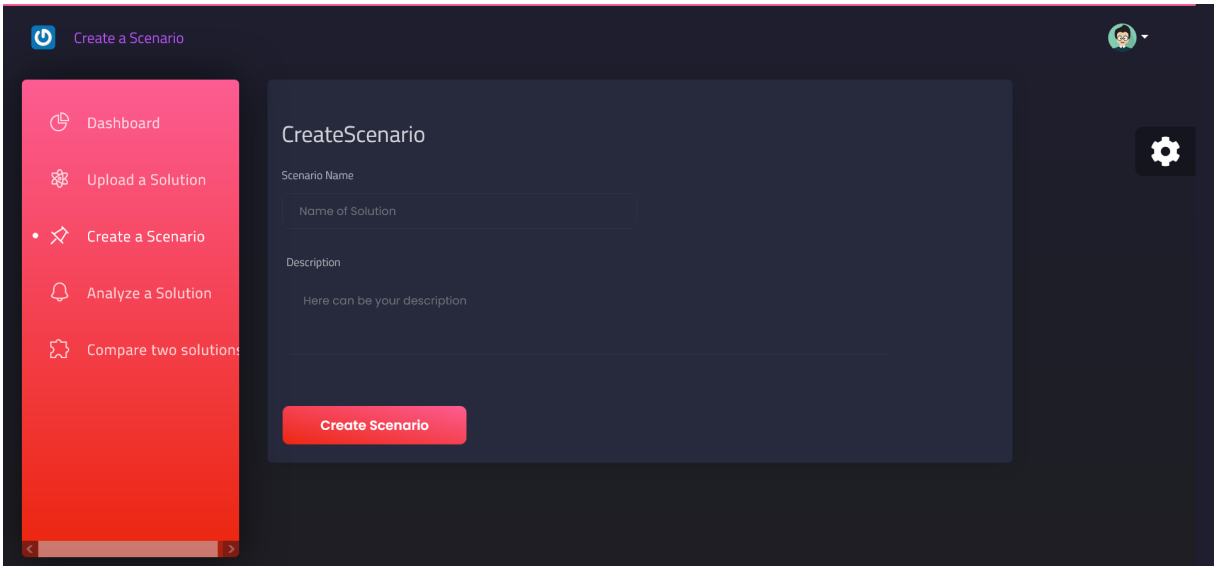

Figure 5.3: Create Scenario Page

section and for unsupervised solutions in the unsupervised solutions sections along with the scenario name, solution name in the scenario, solution list section.

A click on the analyze page in the sidebar redirects the user there where he can select a scenario and a associated solution he created. A click on the unsupervised/supervised button displays either the metric scores for supervised or unsupervised solutions after click on analyze the scors are printed on the page

A click on the compare page in the sidebar redirects the user to the compare page where he can select a scenario and two to the scenario linked solutions. The superviesd/unsupervised button either displays the pillar scores properties or the pillarscores.

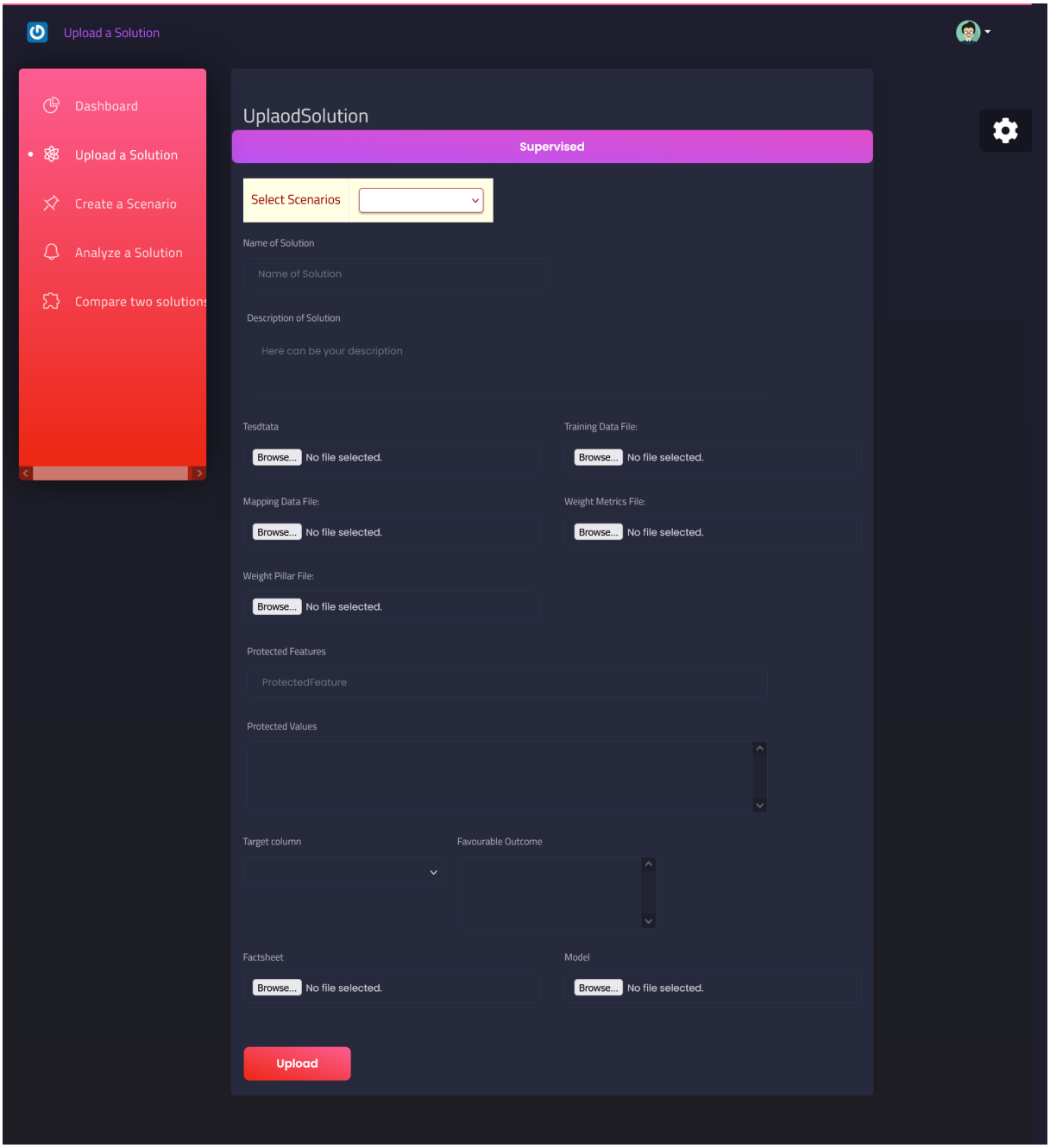

Figure 5.4: Upload Supervised Solution Page

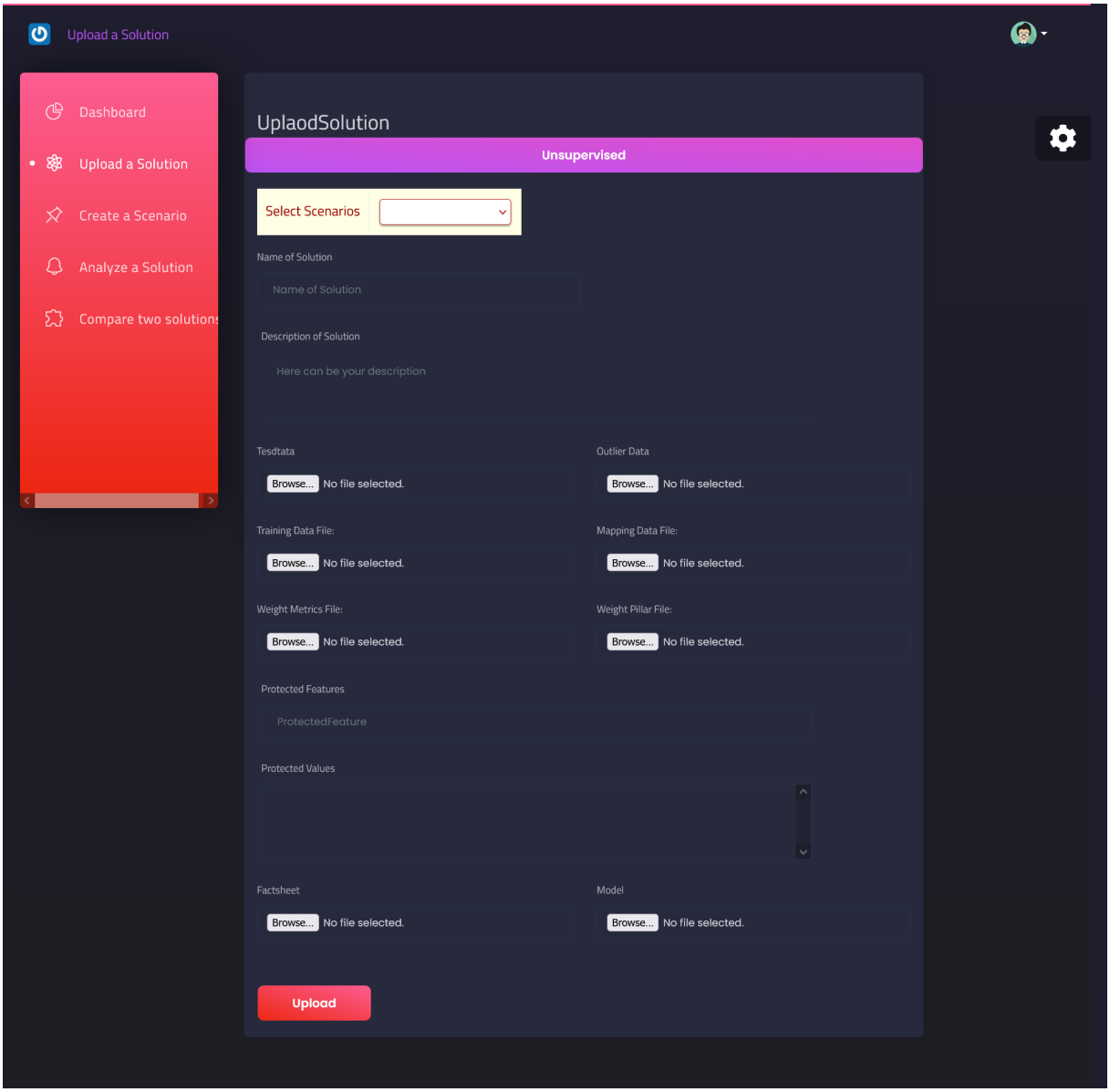

Figure 5.5: Upload Unsupervised Solution Page

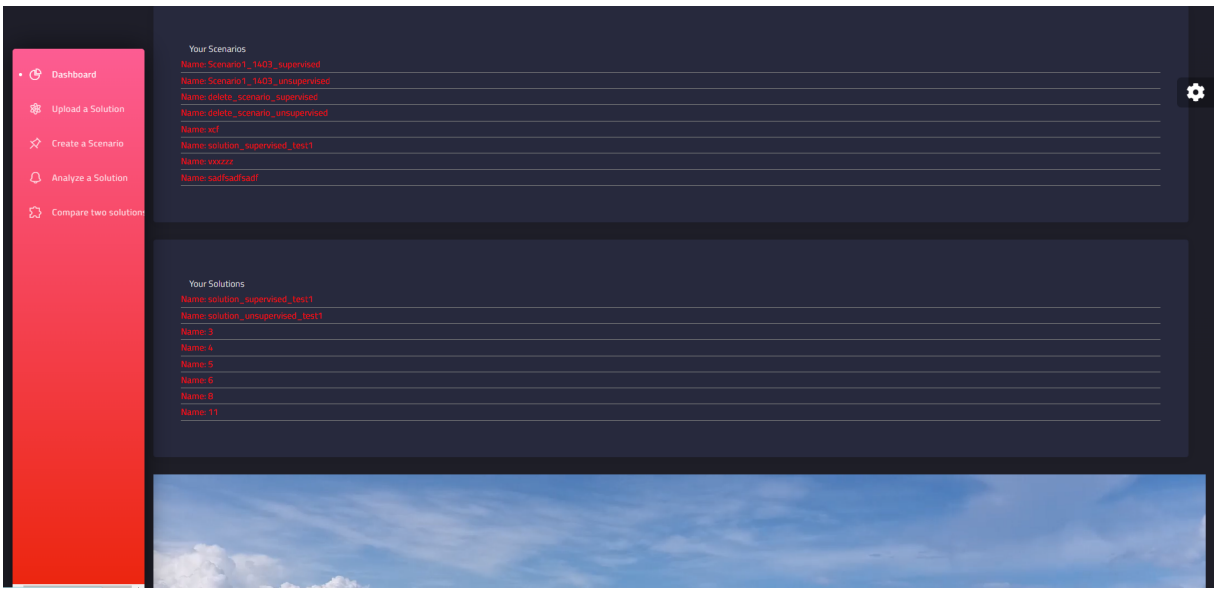

Figure 5.6: Upload Unsupervised Solution Page

Figure 5.7: Upload Unsupervised Solution Page

Chapter 6

Evaluation

6.1 GUI

The scenario, solution creation, edition, deletion, the analyze, compare functionalities for solutions and the dashboard were tested with testvalues sets used in the webapps of the two previous iterations.

6.1.1 Testvalues

To evaluate the webapp functionalities for unsupervised Solutions the Cluster-Based Local Outlier Factor algorithm and Copula-Based Outlier Detection algorithm test values sets consisting of training-,test-,outliersdataset and a model and factsheet file were used alongside json files for metrics mapping to scores 1-5, metric weights for calculating the pillar scores and pillar weights for determining the trust score was used.

To evaluate the webapp functionalities for supervised Solutions the Jans SGD Classifier algorithm and Jans Support Vector Machine 01 algorithm test values sets consisting of training-,test-,outliersdataset and a model and factsheet file were used alongside json files for metrics mapping to scores 1-5, metric weights for calculating the pillar scores and pillar weights for determining the trust score were used.

6.1.2 Results

1. Scenario Creation The user can create a scenario by adding giving it a name and a description on the scenario creation page.

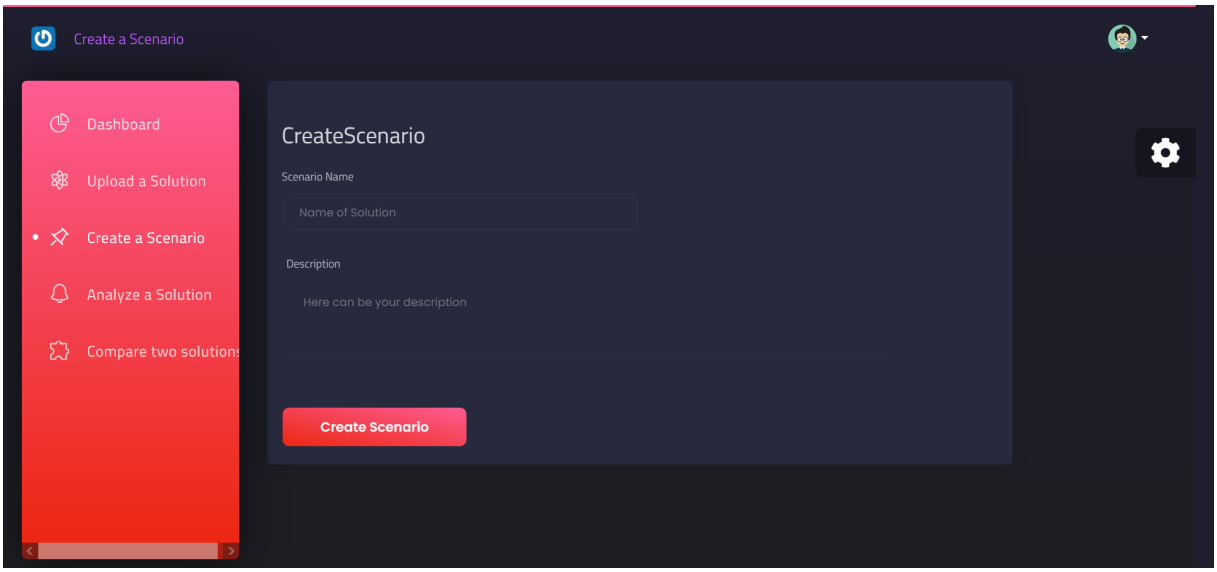

Figure 6.1: Scenario Creation

2. Scenario Detail Page

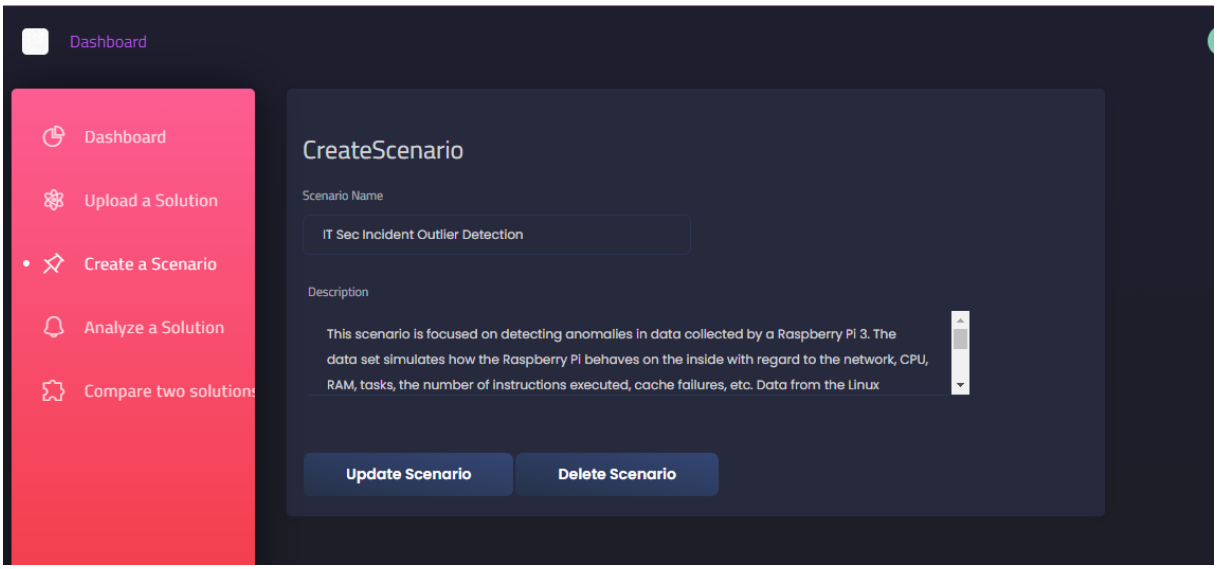

Figure 6.2: Scenario Detail Page

3. Solution Creation The user can create a solution by adding filling out or adding the solution parameter files in the solution upload page.

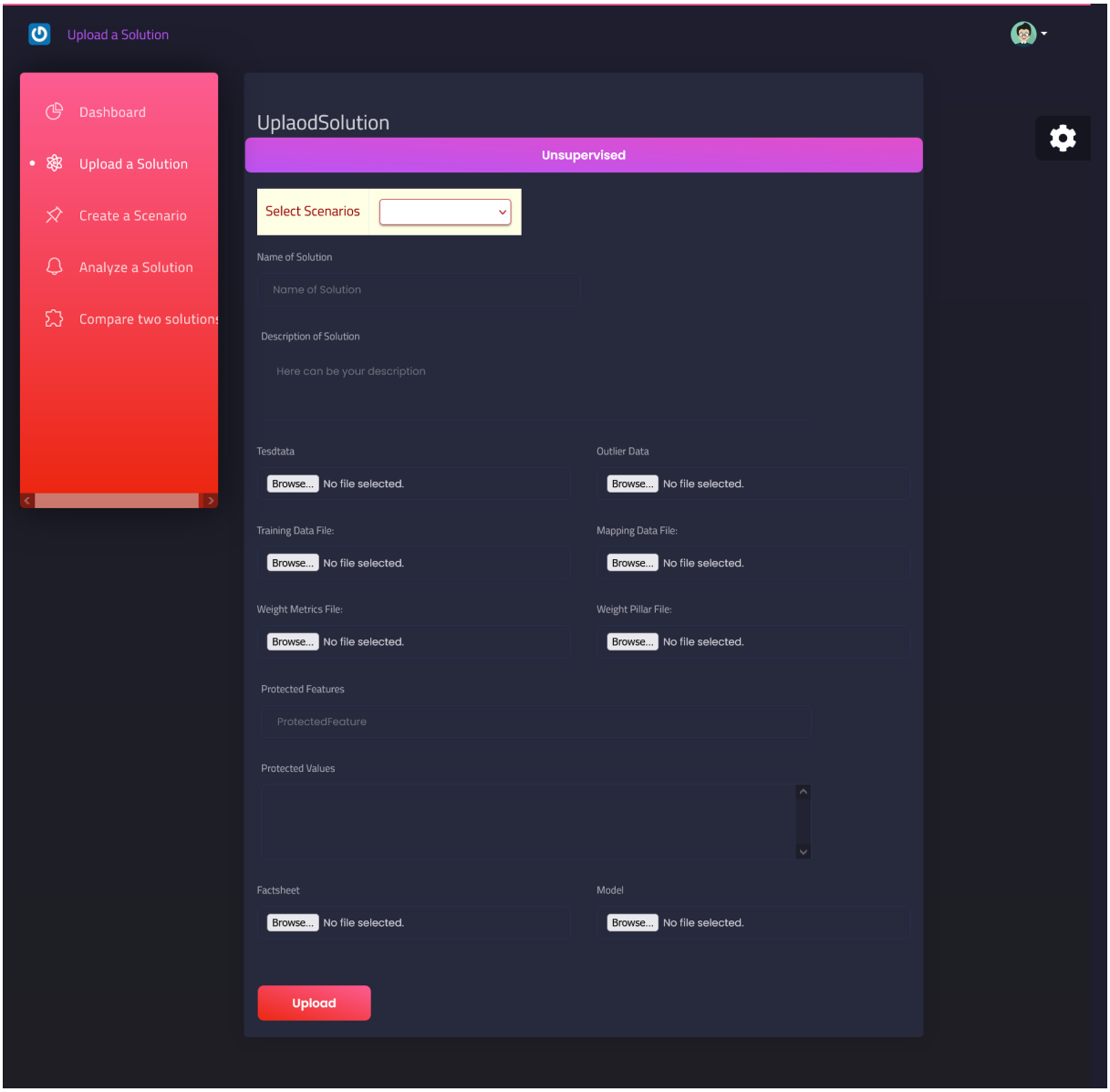

Figure 6.3: Solution Creation

4. Solution Detail Page

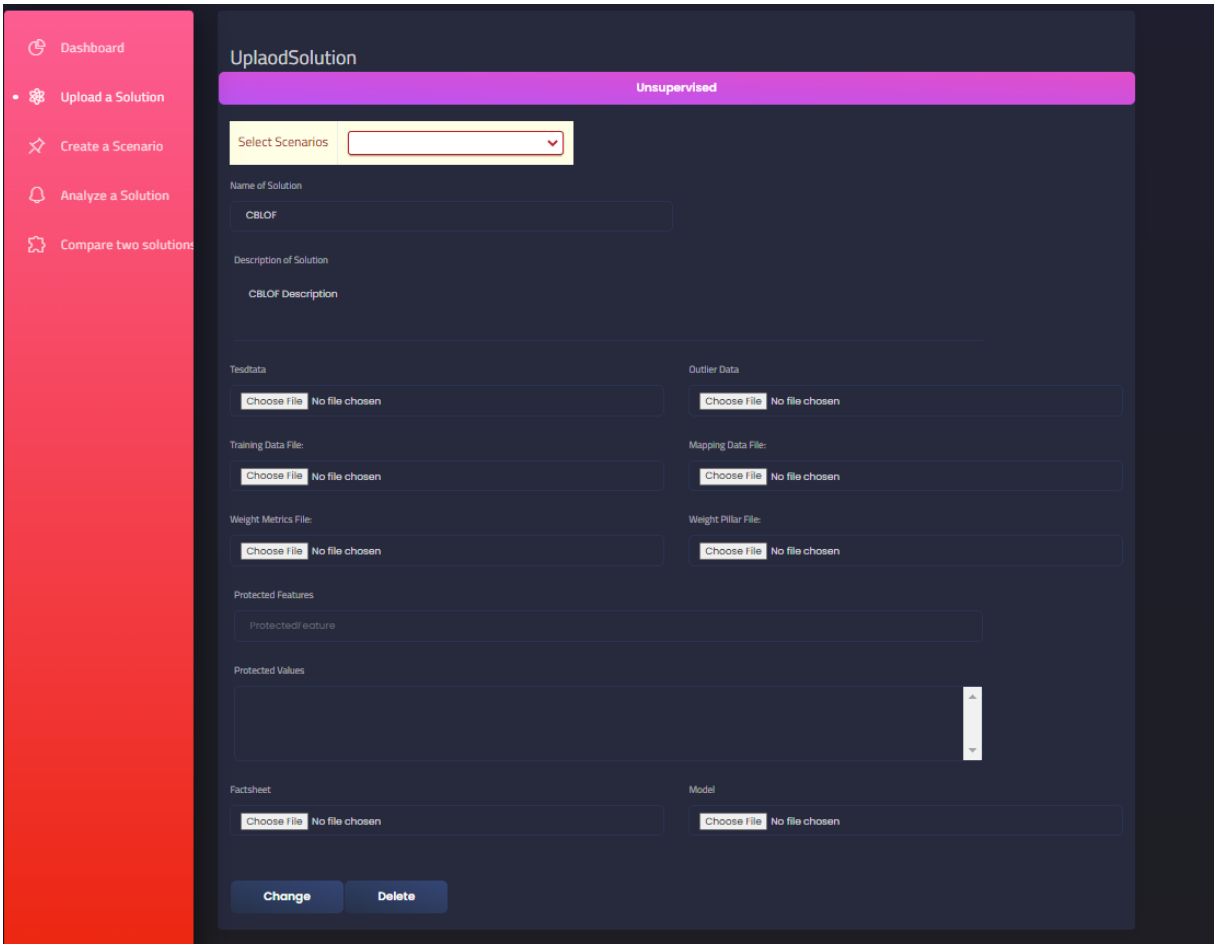

Figure 6.4: Solution detail page

5. Anlayze Solution

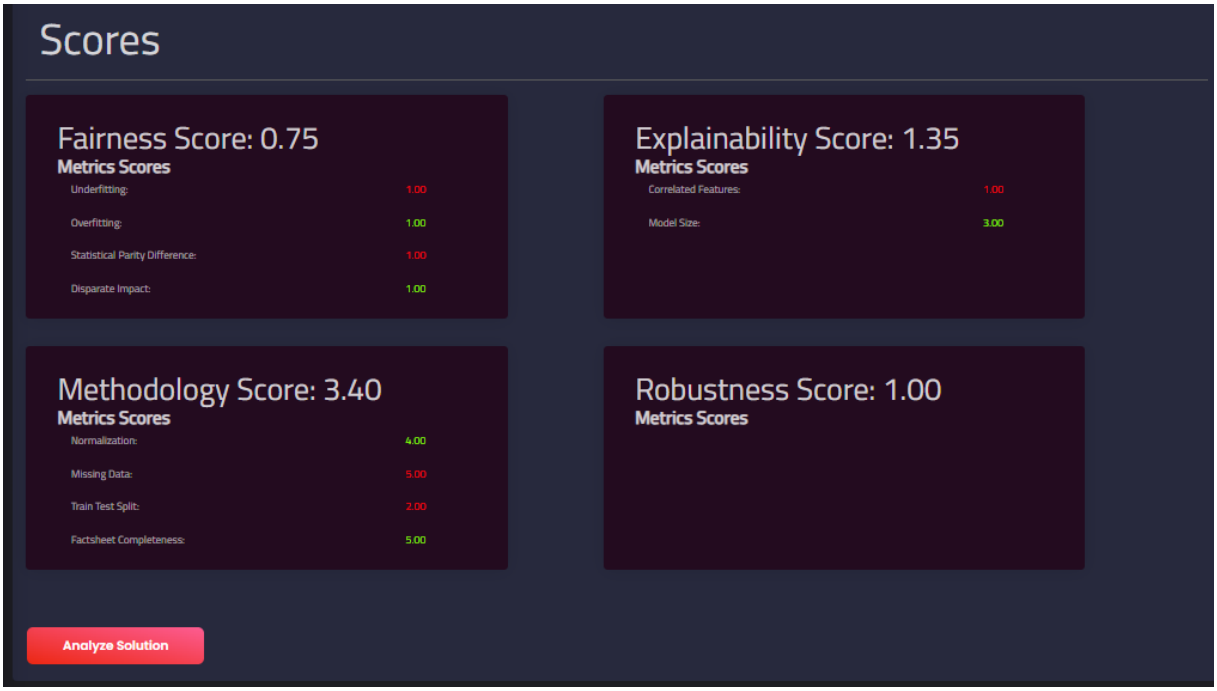

Figure 6.5: Analyze Solution

6. Compare Solutions

Figure 6.6: Compare Solutions

6.2 Postman

The user can send requests to the API endpoints of the server e.g. https://trustcalc-server-dot-ritual-client.oa.r.appspot.com/api/scenario/to create a scenario.

POST /api /scenario

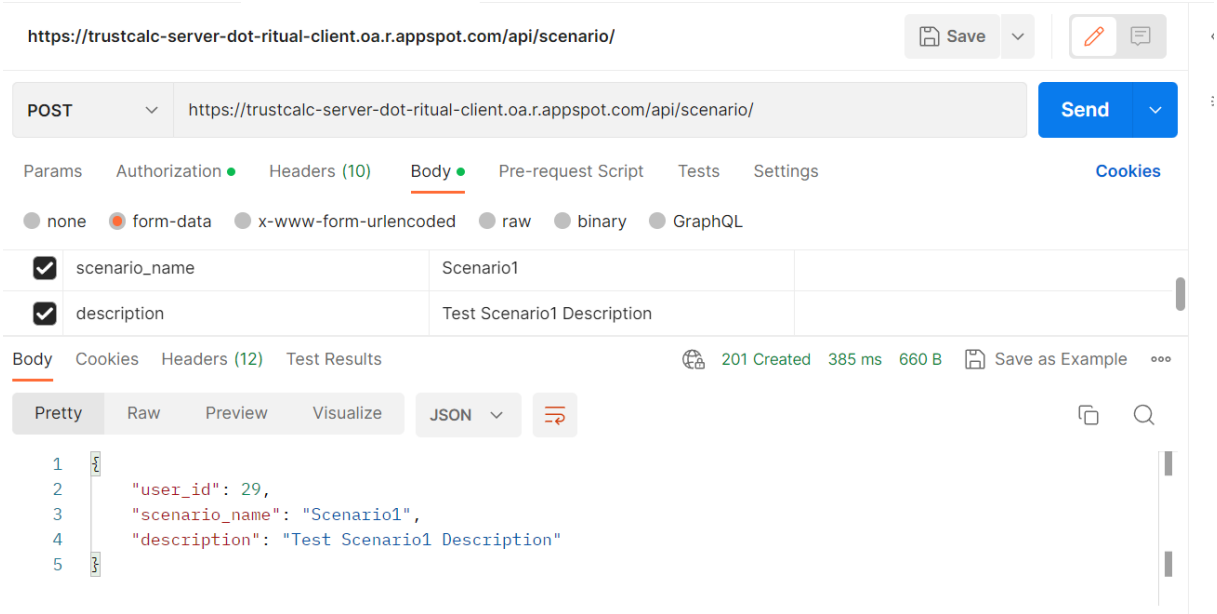

Figure 6.7: Postman

6.3 Performance

The performance of the locally hosted webapp was evaluated using Lighthouse. Lighthouse is a widly used open-source tool developed by Google that can be used to audit web pages for performance, accessibility, best practices, and search engine optimization. Following it is used as a tool for evaluating the quality and performance of web applications[17].

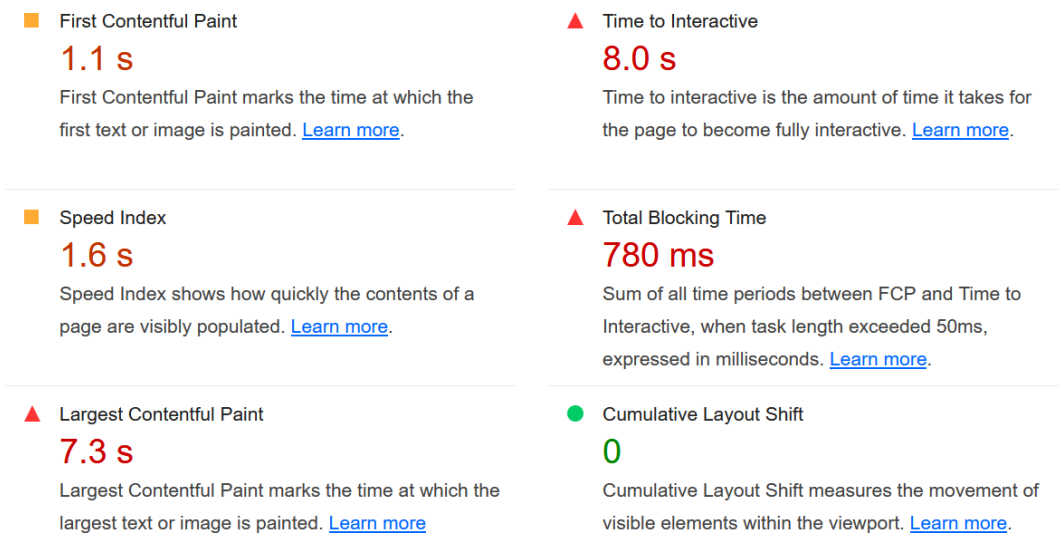

Figure 6.8: Performance Localhost

6.4 Security

HTTP headers can be ussed to transfer additonal infromation between client and server during a HTTP request/response transaction. Proper configuration is needed to protect against common securitythreats as cross-site scripting (XSS) and cross-site request forgery (CSRF). In the context the security of the HTTP headers w ereverified using the https://securityheaders.com website, which analyzes the HTTP headers of a given web page and provides a report that identifies any security issues or areas for improvement.

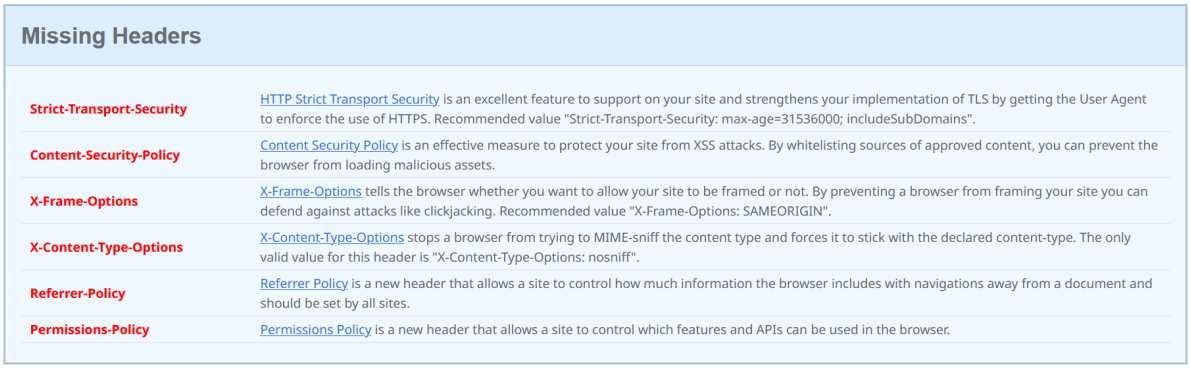

Figure 6.9: Security HTTP headers

The SSL/TLS protocol establishes secure and encrypted connection between a web server, The SSL/TLS configuration of the web application was evaluated using the website https://www.ssllabs.com/.

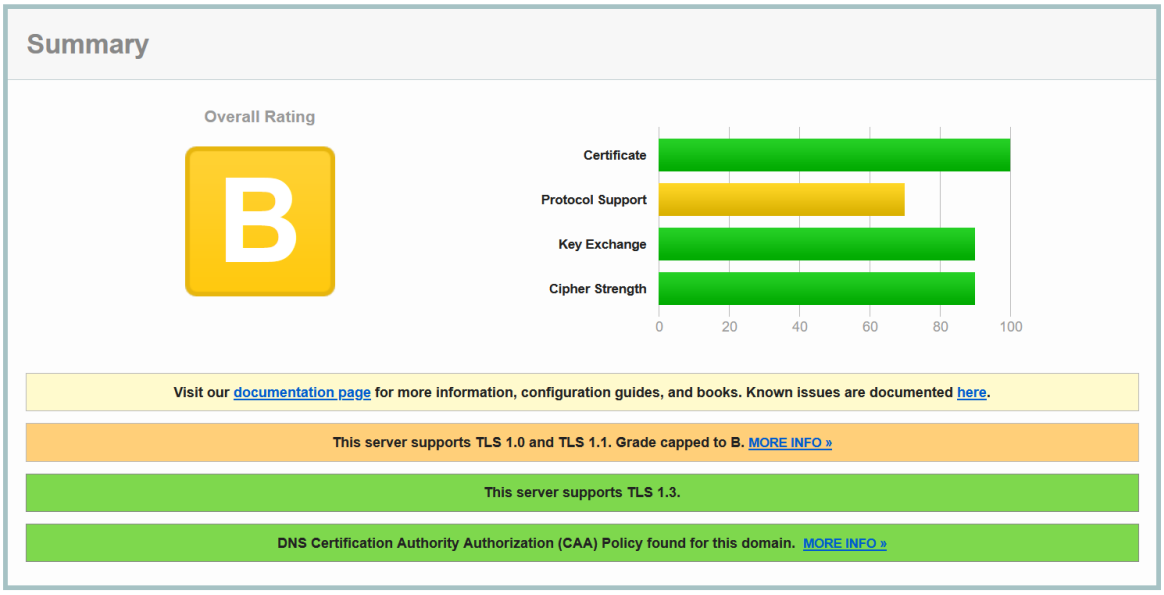

Figure 6.10: Security SSL/TLS configuraiton

6.5 Accessibility

The accessibility of the locally hosted webapp was evaluated using Lighthouse.

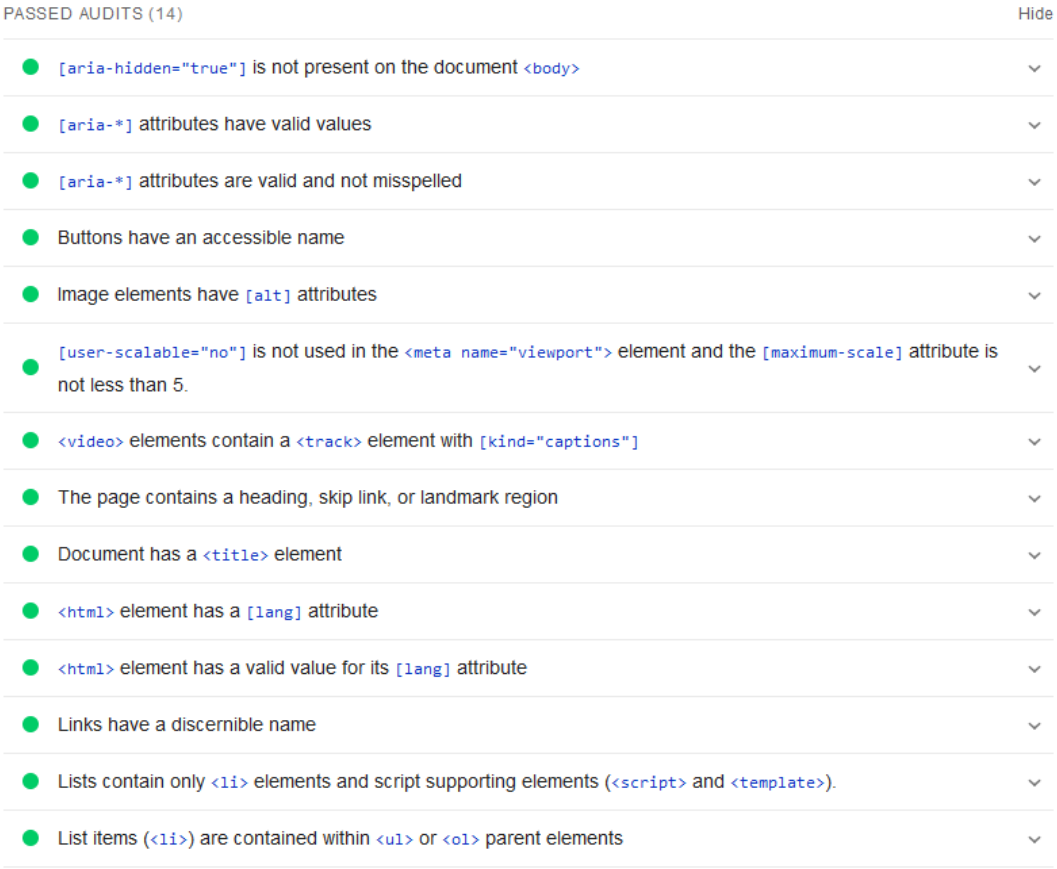

Figure 6.11: Accessibility localhost passed

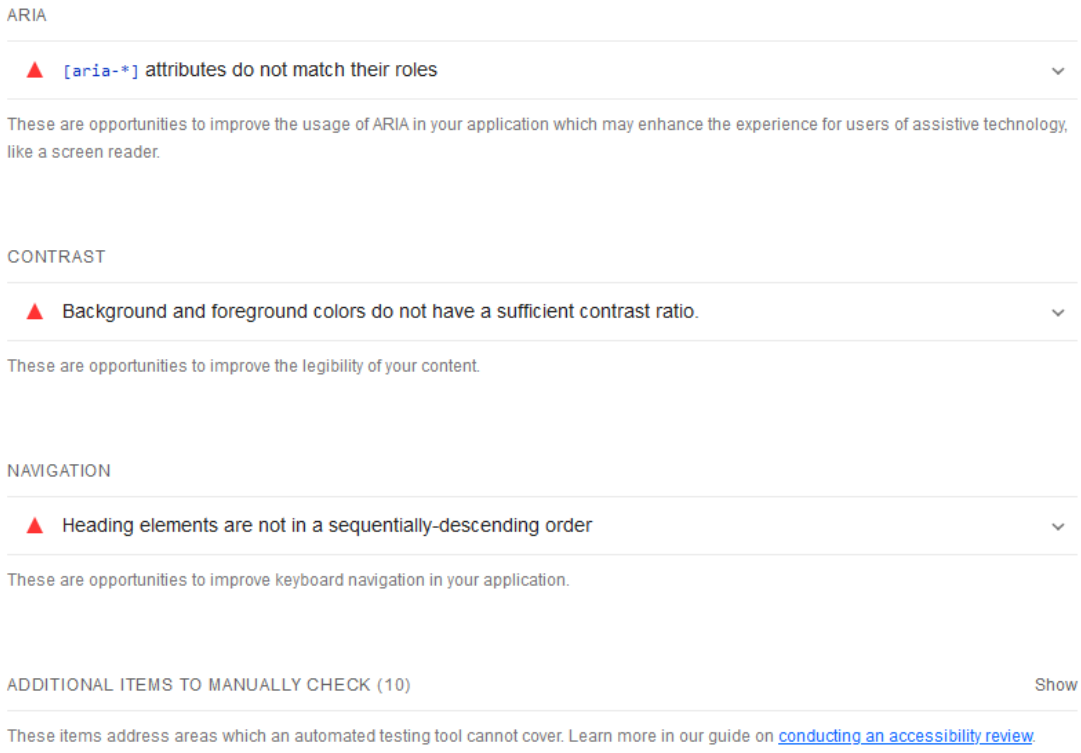

Figure 6.12: Accessibility localhost not passed

Chapter 7

Summary and Conclusions

In the first part of the thesis existing trusted AI platforms for the computaion of the trustowrthiness of supervised and unsupervised ML / DL models were surveyed. In a next step the requirements for the webapp were definied, those included user management, scenario, solution creation, deletion via APIs and the calculation of trustworthiness levels and considerations of security aspects. The requirements were than used as basis for teh evealuation of different web technologies and the decision to use angulalar as frontend and django as backend framework. The design of the webapp was prepared, by reasearching different solutions for the requirements, considering the framework specific characteristics and genereting first solutions sketches. The webapp was then accordingly implemented. The implementation was during this process tested using testvalues from previous works, and then deployed on ritual-ai.net.

7.1 Future Work

The current webapp usability could be enhanced, through better frontend design, the performance speed can be improved by choosing a different deployment method, The functions for determinig the different metric, pillar scores should be rewritten to make them less error prone and the process of saving trained model files and loading them should be generalized, and factsheets should have a uniform appearance.

[german]

Bibliography

[1] Florian TramÃ¨r et al. *FairTest: Discovering Unwarranted Associations in Data-Driven Applications*. arXiv:1510.02377 [cs]. Aug. 2016. URL: http://arxiv.org/abs/1510.02377 (visited on 02/15/2023).

[2] Deepak Muralidharan et al. *Leveraging User Engagement Signals For Entity Labeling in a Virtual Assistant*. arXiv:1909.09143 [cs, stat]. Sept. 2019. DOI: 10.48550/arXiv.1909.09143. URL: http://arxiv.org/abs/1909.09143 (visited on 02/15/2023).

[3] Maria-Irina Nicolae et al. *Adversarial Robustness Toolbox v1.0.0*. arXiv:1807.01069 [cs, stat]. Nov. 2019. URL: http://arxiv.org/abs/1807.01069 (visited on 03/25/2023).

[4] Vijay Arya et al. *AI Explainability 360: Impact and Design*. arXiv:2109.12151 [cs]. Sept. 2021. DOI: 10.48550/arXiv.2109.12151. URL: http://arxiv.org/abs/2109.12151 (visited on 02/15/2023).

[5] Haochen Liu et al. *Trustworthy AI: A Computational Perspective*. arXiv:2107.06641 [cs]. Aug. 2021. URL: http://arxiv.org/abs/2107.06641 (visited on 02/15/2023).

[6] Alberto Huertas Celdran et al. "RITUAL: a Platform Quantifying the Trustworthiness of Supervised Machine Learning". en. In: *2022 18th International Conference on Network and Service Management (CNSM)*. Thessaloniki, Greece: IEEE, Oct. 2022, pp. 364–366. ISBN: 978-3-903176-51-5. DOI: 10.23919/CNSM55787.2022.9965139. URL: https://ieeexplore.ieee.org/document/9965139/ (visited on 02/15/2023).

[7] Zhenpeng Chen et al. *Fairness Testing: A Comprehensive Survey and Analysis of Trends*. arXiv:2207.10223 [cs]. Aug. 2022. URL: http://arxiv.org/abs/2207.10223 (visited on 02/15/2023).

[8] *Django introduction - Learn web development | MDN*. en-US. Feb. 2023. URL: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction (visited on 03/25/2023).

[9] *1.Â Introduction to Spring Framework*. URL: https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/overview.html (visited on 03/25/2023).

[10] *Angular - PreloadingStrategy*. URL: https://angular.io/api/router/PreloadingStrategy (visited on 03/25/2023).

[11] *Django*. en. URL: https://docs.djangoproject.com/en/4.1/topics/cache/ (visited on 03/25/2023).

[12] *FAT Forensics â FAT Forensics 0.1.2 documentation*. URL: https://fat-forensics.org/ (visited on 03/25/2023).

[13] *FICOÂ® FalconÂ® Fraud Manager*. en. URL: https://www.fico.com/en/products/fico-falcon-fraud-manager (visited on 02/15/2023).

[14] *Introducing ChatGPT*. URL: https://openai.com/blog/chatgpt (visited on 03/25/2023).

[15] *Introduction | Vue.js*. URL: https://vuejs.org/guide/introduction.html (visited on 03/25/2023).

[16] *Laravel - The PHP Framework For Web Artisans*. en. URL: https://laravel.com/ (visited on 03/25/2023).

[17] *Lighthouse overview*. en. URL: https://developer.chrome.com/docs/lighthouse/overview/ (visited on 03/25/2023).

[18] *React â A JavaScript library for building user interfaces*. en. URL: https://reactjs.org/ (visited on 03/25/2023).

[19] *Welcome to the Adversarial Robustness Toolbox â Adversarial Robustness Toolbox 1.14.0 documentation*. URL: https://adversarial-robustness-toolbox.readthedocs.io/en/latest/ (visited on 03/25/2023).

[20] *What is django ORM*. URL: https://www.tutorialspoint.com/what-is-django-orm (visited on 03/25/2023).

List of Figures

List of Tables