A Showcase for the
Titan Control Center

Bachelor’s Thesis

Julian Grabitzky

March 30, 2021

KIEL UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE
SOFTWARE ENGINEERING GROUP

Advised by: Prof. Dr. Wilhelm Hasselbring
Sören Henning, M. Sc.
Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Kiel, 30. März 2021

iii
Abstract

In times with an ever-increasing amount of generated data by Industrial Internet of Things sensors which needs to be collected and processed in near real-time. This leads to the need for an analytic platform with sophisticated software systems to process Big Data in near real-time.

The Titan project is an example of a project which implements the Industrial DevOps approach to solve this challenge. Within the Titan platform, the Titan Control Center is a scalable architecture to analyze data streams in near real-time.

This thesis aims to create a showcase for the Titan Control Center with a usability evaluation at the end. The showcase covers each key functionality of the Titan Control Center. The thesis starts with developing a new data generator to generate realistic measurements for the showcase. Additionally, we extend the Titan Control Center’s current visualizations to provide the user with simple visualizations to gain the needed information from the monitored data quickly. After that, we conduct a usability evaluation of the system and discuss the results afterward.
# Contents

1 Introduction 1

1.1 Motivation .......................................................... 1

1.2 Goals ........................................................................... 1

1.2.1 G1: Developing a Data Generator based on the HIPE Data Set ........................................ 2

1.2.2 G2: Implementing the Visualizations for Forecasting .......................................................... 2

1.2.3 G3: Extending the Visualization for Anomaly Detection ...................................................... 2

1.2.4 G4: Extending the Titan Control Center Visualizations ......................................................... 2

1.2.5 G5: Evaluating the Titan Control Center .............................................................................. 2

1.3 Document Structure ......................................................... 3

2 Foundations and Technologies 5

2.1 Industrial DevOps .......................................................... 5

2.2 The Industrial DevOps Platform Titan ..................................................................................... 5

2.2.1 Titan Flow Engine .................................................... 6

2.2.2 Titan Control Center ................................................ 7

2.2.3 Titan Control Center Frontend ..................................... 9

2.3 The Containerization Technology Docker .............................................................................. 11

2.4 The Machine Learning Library Scikit-learn ......................................................................... 12

2.5 The Distributed Streaming Platform Apache Kafka .................................................................. 12

2.6 The Web Framework Vue.js .................................................................................................... 12

2.7 The Programming Language Typescript ................................................................................. 13

2.8 The Visualization Library Britecharts .................................................................................... 13

2.9 The HIPE Data Set .................................................................................................................... 13

3 Approach 15

3.1 Developing the the Data Generator ....................................................................................... 15

3.1.1 Creation of the Scikit-learn Models ...................................................................................... 15

3.1.2 Implementation with the Titan Flow Engine ......................................................................... 16

3.2 Creating a new UI Component for Forecasting ...................................................................... 18

3.3 Extending the UI Component for Anomaly Detection ............................................................. 19

3.4 Extend the Visualizations ...................................................................................................... 20

4 Usability Evaluation 23

4.1 Goals ............................................................................. 23

4.2 Methodology .................................................................. 23

4.3 Experimental Setup .......................................................... 24
Contents

4.4 Results ......................................................... 26
4.5 Discussion ..................................................... 27
4.6 Threats to Validity .............................................. 29

5 Related Work .................................................. 31

6 Conclusion and Future Work .................................... 33
  6.1 Conclusion ..................................................... 33
  6.2 Future Work ................................................... 34

Bibliography ....................................................... 35
Chapter 1

Introduction

1.1 Motivation

The Titan project transfers the known DevOps approach to industrial production environments. The applied approach is called Industrial DevOps, and the Titan project is an example for implementing Industrial DevOps [Hasselbring et al. 2019]. The fundamental concept of Industrial DevOps is a constant process of operation, observation, and development of the whole production environment.

Nowadays, the number of industrial components which can monitor their state is increasing. This results in a vast amount of generated data from the Industrial Internet of Things sensors (IIoT). With this new opportunity, industrial manufactures can automatically collect, analyze, and visualize data. This new trend is often referred to as Industry 4.0. But many small enterprises do not utilize Industry 4.0 solutions because this requires a major investment in several areas of the enterprise.

Within the Titan project, there is the Titan Control Center, a scalable software platform to analyze data streams from the IIoT sensors in near real-time [Henning et al. 2021]. Besides, the Titan Control Centre provides powerful tools to combine and display data.

This thesis aims to create a showcase for the Titan Control Center by creating a new data generator and improving the current visualization capabilities. We added a new frontend component for the existing Forecast microservice to display each machine sensors future prediction. Furthermore, we extend the Anomaly Detection microservices visualizations by adding a new component to test the microservice by creating artificial anomalies. Additionally, we create new graphs to display monthly and yearly statics for the selected sensor and a stacked area chart to display the partial power consumption of each sensor. Every new visualization is added to the existing single-page application of the Titan Control Center, which is built with the Framework Vue.js [You 2021].

1.2 Goals

Our overall goal for this thesis is to build a showcase for the Titan Control Center to present each key functionality. To achieve this, we divide the Titan Control Center into each key component to extend and evaluate it.
1. Introduction

1.2.1 G1: Developing a Data Generator based on the HIPE Data Set

Our first goal is to build a new data generator to generate power consumption data for the Titan Control Center, which can be used in live demos or to test new features. This provides the user with a reliable and realistic source of test data. The data generator is implemented with the Titan Flow Engine and is based on historical data from the HIPE data set [Bischof et al. 2018]. The HIPE data set contains the power consumption measurements of a small production site with various machines. This leads to a more realistic power consumption pattern for our data generator than the current data generator.

1.2.2 G2: Implementing the Visualizations for Forecasting

The second goal is to create a new user interface component for the Control Center’s Forecast microservice [Henning et al. 2021]. This allows the user to see and analyze the predicted power consumption of the production environment to detect possible spikes and prepare countermeasures.

1.2.3 G3: Extending the Visualization for Anomaly Detection

Our third goal is to extend the user interface component for the Control Center’s Anomaly Detection microservice, which will enable the user to quickly detect anomalies in the monitored system [Henning et al. 2021]. Also, the user will be able to configure the visualization for the Anomaly Detection service. For example, the user will adjust the anomaly score, which is used to determine if the corresponding data point should be interpreted as an anomaly. Furthermore, we will create another user interface component called Anomaly Generator to create artificial anomalies to test the system.

1.2.4 G4: Extending the Titan Control Center Visualizations

The fourth goal is to create new visualizations for the Titan Control Center, which provide a rich user experience. To gain knowledge of the vast amount of visualized data, clear and simple visualizations are needed so that the user is not overwhelmed with unnecessary information. The first visualization is a new stacked area chart to display the partial power consumption of each sensor. Additionally, we create two new bar charts to display monthly and yearly power consumption statics for each sensor.

1.2.5 G5: Evaluating the Titan Control Center

An evaluation of the Titan Control Center’s whole front-end components is the last goal of this thesis. In the evaluation, we analyze the Titan Control Center’s usability with one test person with prior experience in using the Titan platform. The focus of the evaluation tasks is based on five essential usability characteristics [Holzinger 2005].
1.3 Document Structure

The remainder of this thesis is structured as follows. Chapter 2 introduces essential foundations and technologies to build a foundation. Then, we describe how we integrate our approach into the Titan Control Center in Chapter 3. In Chapter 4, we describe the evaluation method and setup. Afterward, we evaluate and discuss the results of the evaluation. In Chapter 5, we discuss related work and Chapter 6 concludes this thesis and presents an outlook regarding future work.
In this chapter, we will outline and explain the general foundations and used technologies required for the thesis. Firstly, we explain the terminology of Industrial DevOps. After that, we break down the Industrial DevOps platform Titan and explain each component. Additionally, we point out the underlying technologies of the components.

2.1 Industrial DevOps

The development of Software is traditionally separated from the operation, but this impedes efficient communication, collaboration and integration. To solve these challenges, the concept of DevOps introduces a closer collaboration between the development and operation of the given software development [Lwakatare et al. 2015].

Industrial DevOps is an approach to transfer the culture and methods of DevOps to industrial production environments. The core concept of this is to embed a new cyclic, continuous adaptation and improvement process to the existing production environment (see Figure 2.1). This process starts with monitoring and analyzing the current system during its operation. Based on the results of the analyzed system and possible new requirements, a new adaptation plan is identified. These changes are implemented, and the process is monitored again. Crucial to the concept is the continuous process to react flexibly to new requirements [Hasselbring et al. 2019].

2.2 The Industrial DevOps Platform Titan

The Titan project is a research project to investigate possible solutions to digitize production environments. It is a software platform to monitor and integration existing industrial production environments based on Industrial DevOps [Hasselbring et al. 2019]. It offers dedicated components to collect, display, and analyze data in real-time. Besides that, the system uses flow-based programming [Morrison 2010] principles to adapt the system. This solves the system’s dependency to need a dedicated software developer to adapt and change the running system.
2. Foundations and Technologies

![Diagram](image)

**Figure 2.1.** The continuous adaptation and improvement process of Industrial DevOps [Hasselbring et al. 2019]

### 2.2.1 Titan Flow Engine

The Titan Flow Engine is a unique software component to integrate power consumption data into the Titan Control Center [Hasselbring et al. 2019]. This provides the functionality to enrich the data by format and unit conversion, filtering, and aggregation. It provides a visual modeling interface for the data flows in the industrial production environment, based on the principles of flow-based programming [Morrison 2010].

Every component in the data flow is a so-called brick that can perform data collection, transformation, or creation. Also, each brick has an input and output port, which are connected to different bricks. One or more connected bricks are a so-called flow. This enables the operation team of the industrial production environment to reconfigure the power consumption data flow, so that a new sensor can be added without the need of a dedicated software developer with advanced programming skills. The Titan Flow Engine provides a variety of different brick types for the user (see Figure 2.2).

**Inlet** The Inlet brick is the starting point of each flow (see Figure 2.2 Nr. 1). This brick has no input port, but at least one output port connects to the next brick. Furthermore, this brick is used to model machine sensors or generally to ingest data into the system.
2.2. The Industrial DevOps Platform Titan

**General**  The General brick is used when the tasks do not fit into one of the other bricks (see Figure 2.2 Nr. 2). It is a general base brick with no specified functionality. Furthermore, it has at least one input and output port to connect to the next brick in the flow.

**Filter**  The next brick is the Filter brick which filters the incoming data stream based on the defined condition (see Figure 2.2 Nr. 3). This brick has at least one input and output port.

**Selector**  The Selector brick manages the flow direction of the incoming data stream (see Figure 2.2 Nr. 4). It has at least one input port for the incoming data and at least two output ports. Furthermore, the Selector bricks route the incoming data based on the predefined condition to the correct output port.

**Outlet**  The Outlet brick is the last brick of a flow, and this connects the flow to different external components such as databases (see Figure 2.2 Nr. 5). The Outlet is defined with at least one input port for the incoming data.

### 2.2.2 Titan Control Center

Within the Titan research project, the Titan Control Center is a scalable implementation of a system that offers components to collect, display, and analyze power consumption data from various sources within the industrial production environment [Henning and Hasselbring 2021]. The architecture of the Titan Control Center is presented in Figure 2.3. The system takes advantage of the microservice pattern to separate different tasks into loosely-coupled microservices [Hasselbring and Steinacker 2017]. Each of the loosely coupled microservice is responsible for a different task and runs in a different isolated container with its own state. Now we are going to explain the area of responsibility of each microservice.

**Integration Edge Component**  The power consumption data starts in the integration edge component where the sensors are grouped and publish their data to the asynchronous message broker Kafka [Kreps 2011]. The Titan Flow Engine from Section 2.2.1 is an example for a integration of edge components.
Aggregation Service  The Aggregation microservice collects the power consumption data of the various sensors from the corresponding Kafka Topic and provides aggregated data for the other microservice [Henning and Hasselbring 2020].

History Service  The History microservice stores normal and aggregated data in the given database and provides different API endpoints for the other services to access the stored data. For example, an API endpoint to obtain the power consumption data for a single sensor or a group of sensors is implemented. Besides that, there are API endpoints to access aggregated minutely and hourly power consumption data [Henning et al. 2021].

Statistics Service  On the other hand, the Statistic service provides statistical data like trends.

Anomaly Detection Service  The Anomaly Detection microservice detects anomalies in the power consumption data and provides an API endpoint to access the detected anomalies.

Forecasting Service  The next microservice is the Forecasting service, which provides a forecast for the sensor measurements [Boguhn 2020].
2.2. The Industrial DevOps Platform Titan

Sensor Management Service  The Sensor Management manages the hierarchical model that specifies which sensor exists.

Control Center  The Control Center provides a user interface to access all functionalities of the other microservices of the Titan Control Center. It serves a single-page application which is implemented with Vue.js [You 2021] and Britecharts [Eventbrite 2021]. Additionally, a more detailed explanation of the Control Center follows in Section 2.2.3.

Grafana Dashboard  The Grafana Dashboard is the second visualization of the Titan Control Center, which is implemented with Grafana [Grafana Labs 2021] (see Figure 2.4). This dashboard provides the user with a broad set of common visualization, e.g., line charts and bar charts. Besides that, the dashboard is highly customizable so that the users can add, modify or create a new custom dashboard for a subset of users. In contrast to the Control Center, the dashboard does not provide complex interactive visualizations or management functionality (e.g., sensor management).

2.2.3 Titan Control Center Frontend

The Control Center is a user interface to access all functionalities of the other implemented microservices. Figure 2.5 shows a screenshot of the current Titan Control Center Dashboard. It consists of several components for each functionality/microservice, accessible over the navigation bar on the left side. On the top right side is an embedded link to the Titan Flow Engine and a time selector to access and inspect historical data.

Sensor Details  The Sensor Details view enables the user to inspect a sensor or sensor group configured under the Sensor Management view. Besides, this view displays the current power consumption trend and a line chart for the active power consumption. Following that, this view also provides a histogram (e.g., to detect possible load peaks) and a composition chart to check the given sensor’s partial power consumption in contrast to the next higher sensor group. The next functionality are two graphs for the daily and weekly course, which display the aggregated power consumption of the last weeks/months to identify power consumption patterns. Moreover, the weekly and daily consumption heatmaps are alternative graphs to the daily and weekly course.

Comparison  The Comparison view enables the user to compare the power consumption of different sensor or sensor groups. This is displayed with a line chart, and new sensors are added to the existing line chart. In this way, the user can explore the correlation between sensors.

Anomalies  The Anomalies view provides the user with an interface to detect possible anomalies in the machine sensors’ raw and aggregated data. In this view, the user can choose which sensor to inspect. The user interface provides a line chart with the cur-
2. Foundations and Technologies

Figure 2.4. Screenshot of the Dashboard [Wetzel 2019]

rent power consumption and a table that lists possible anomalies with the corresponding anomaly score. Besides, the user can configure the anomaly score to refine the listing.

**Sensor Management**  The Sensor Management view provides the user with the necessary tooling to configure the sensor hierarchy. Existing sensors can be renamed, and unused sensors are listed at the bottom. Also, the user can create new aggregated sensors.
2.3 The Containerization Technology Docker

Containerization [Bernstein 2014] is a technique to virtualize the software’s execution environment, which means that each container is isolated from the host and other running containers. This results in a lightweight virtualization technique that produces less overhead than traditional virtualization techniques such as virtual machines.

A common technology for this use case is the open-source platform Docker [Docker 2021]. Each Docker container is created from a specified container image (e.g., Docker image). Every Docker image consists of a filesystem with the desired application and the dependencies. These Docker images are created from a Dockerfile, which can include base images to extend their functionality. It is also possible to add files, install software and run commands to start the application. Every service in the Titan Control Center is containerized with Docker.

Figure 2.5. Screenshot of the Control Center
2. Foundations and Technologies

2.4 The Machine Learning Library Scikit-learn

Scikit-learn is a high-level machine learning library with simple and efficient tools to create applications with machine learning [Pedregosa et al. 2011]. The library uses the rich environment of many state-of-the-art implementations of machine learning algorithms for supervised and unsupervised machine learning. At the same time, it maintains a simple interface for the programming language Python. We will use this library to create the machine learning models for the data generator.

2.5 The Distributed Streaming Platform Apache Kafka

Apache Kafka [Kreps 2011] is a scalable fault-tolerant distributed system to transmit and store messages with high throughput and low latency. It is often used for event streams and big data processing because of the highly scalable and fault-tolerant architecture. The data in Kafka is stored as records in topics, and this topic can be separated into multiple partitions based on the record’s key. Moreover, the partitions can be stored on different nodes to increase the fault-tolerance of the system. Each of the partitions of the given topic is an immutable, ordered sequence of records, where each record contains a key-value pair and a ingest timestamp.

**Kafka Streams** Kafka Streams is a stream processing framework for microservices to leverage the advantages of Kafka. It provides a declarative way to describe processing and transformation steps, and Kafka topics are used as input and output of the processed data. The application framework then generates the necessary consumer, producer, and immediate Kafka topics.

2.6 The Web Framework Vue.js

The current implementation of the Titan Control Center uses the JavaScript framework Vue.js [You 2021]. This is used to develop interactive web interfaces. Vue.js is used to build single-page applications (SPA). These are applications that run in the user’s web browser as a single web page. Also known as client-side renders applications. In contrast to a multi-page application, every new user interaction leads to a newly rendered version of the page.

SPAs in Vue.js are component-based structures with independent logic and appearance. Each component represents an individual part of the user interface. In addition, this encapsulation leads to highly reusable components. Furthermore, components in Vue.js can be composed into larger components.

On the other side, Vue.js forces a loose coupling of components. The data communication and data exchange is only allowed between a parent and a child component.
2.7. The Programming Language TypeScript

Furthermore, communication and data exchange between siblings are prohibited to reduce cyclic dependencies. Lastly, children can only emit an event to their parent, and the event handling is up to the parent component.

2.7 The Programming Language TypeScript

The programming language TypeScript is a typed superset of JavaScript [Bierman et al. 2014]. This results in a refinement of JavaScript by adding static types, classes, and interfaces to JavaScript. Moreover, because of the superset relation between TypeScript and JavaScript, every valid JavaScript program is a valid TypeScript program, leading to a flexible programming environment. In our thesis we use the programming language TypeScript to build our components for the Titan Control Center.

2.8 The Visualization Library Britecharts

The Titan Control Center’s new visualizations are built with the charting library Britecharts [Eventbrite 2021]. Britecharts are client-side rendered charts based on D3.js, which solves the problems of regular D3.js charts, namely, lack of modularity, high complexity, and low re-usability. To achieve modularity and re-usability, the charts are component-based.

2.9 The HIPE Data Set

The HIPE data set is provided by the Institute of Data Processing and Electronics of the Karlsruhe Institute of Technology, which operates a production site for small electronics [Bischof et al. 2018]. This production site produces small electronic systems for particle physics, battery systems, and other areas in small charges, i.e., less than 1000 pieces at a time. Each machine is equipped with a high-resolution smart meter to collect various metricizes. Furthermore, the production site was monitored for three months while the production site produced different products, resulting in a different power consumption pattern per product. Figure 2.6 shows the simultaneous machine activity of each machine in comparison to the other over the three months.
2. Foundations and Technologies

Figure 2.6. The simultaneous machine activity over three months [Bischof et al. 2018]
Chapter 3

Approach

In this chapter we describe the implementation details for each goal corresponding changes to the current codebase of the specific component to achieve this goal.

3.1 Developing the the Data Generator

The first part of our thesis is developing a data generator with the graphical flow modeling tool from the Titan Flow Engine. This flow will consist of several bricks. Firstly we create one brick for each machine sensor of the HIPE data set [Bischof et al. 2018]. These bricks are based on the machines of the HIPE data set and are trained machine learning models, created with Scikit-learn. These models will generate a new measurement for any given date. Furthermore, the models themselves will use a linear regression-based approach to generate the measurements. Additionally, we use the existing CCAdapter brick from the Titan Control Center to publish our generated measurements to the Titan Control Center. This brick publishes the measurements to the corresponding Kafka topic so that the other services can use the incoming data.

3.1.1 Creation of the Scikit-learn Models

Our first step in creating the desired machine learning models starts with the data preparation. The HIPE data set provides various readings for each machine sensor, but we are only interested in the overall power consumption, day of the week, and the hour of the day. We then used the cleaned data to train our linear regression-based model for each machine sensor. The training results in a model, which takes the hour of the day and the day of the week as input and outputs power consumption predictions for the given time. Figure 3.1 shows the prediction of one trained model compared to the actual reading. Some of the trained models showed a scaling problem of the trained model, but the power consumption pattern is relatively accurate to the machine sensor measurements. To compensate for this, we added a configurable scaling factor. Figure 3.2 shows the predicted and scaled power consumption compared to the machine sensor’s actual reading for a given time frame.
3. Approach

Figure 3.1. Prediction of one trained model vs the actual reading

Figure 3.2. Real measurements in comparison to the scaled predictions

3.1.2 Implementation with the Titan Flow Engine

With the trained models for each machine sensor, the next step is to create a brick for each of them. The Titan Flow Engine provides a Python library with abstract brick classes to
3.1. Developing the Data Generator

![Diagram of the complete flow with each machine sensor]

**Figure 3.3.** The complete Flow with each machine sensor

Inherit the base functionality. For our bricks, we use the default InletBrickBase class, which corresponds to the Inlet Brick we described in Section 2.2.1. Each brick is defined by a setup, teardown, and a process method. The setup method is called at the start and the teardown method is executed after the brick receives a stop signal. The process method contains the processing logic of the brick. In our implementation, we loaded the corresponding machine
3. Approach

- Figure 3.4. Screenshot of the Forecast view from the Titan Control Center

model in the setup method. We modified the process model to generate an infinite amount of prediction with a specified interval between the values. Figure 3.3 show the complete flow with every machine sensor of the HIPE data set.

3.2 Creating a new UI Component for Forecasting

This section develops the new UI Component for the Forecasting microservice, which allows the user to see and analyze the predicted power consumption of the industrial production environment and detect possible spikes on a per-sensor basis. The Titan Control Center uses different views, which build upon smaller components to composite more extensive views. Our implementation of the UI component consists of a Forecast view with a corresponding Forecast component. Additionally, we create another component for the line chart from the Britecharts library. The Forecast view includes the Forecast component. The Forecast component provides a sensor selector to switch between the different sensors or sensor groups and it imports the line chart component. This component handles the data transfer with the REST API from the Forecast microservice and the visualization of the predicted data with the reusable line chart from Britecharts. The finished UI component is presented in Figure 3.4.
3.3 Extending the UI Component for Anomaly Detection

The current UI component for the Anomaly Detection microservice is shown in Figure 3.5. It provides the functionality to change the selected sensor and displays the current power consumption in a line chart. Moreover, it provides a table for each occurrence of an anomaly with the corresponding time, value, and anomaly score. Also, the anomaly score is configurable on the top with a default value of three. The anomaly score is an indicator of the intensity of the occurred anomaly.

The first action to extend the current UI for the Anomaly Detection microservice is to add the lower table of the current Anomalies view to the Titan Control Center’s Dashboard to provide another way of accessing the visualization without changing the view. Also, the table displays the detected anomalies of every sensor instead of an individual one.

The next addition to further extend the current UI for the Anomaly Detection microservice is to develop a new view to create artificial anomalies to test the system. For this, we created a new Anomaly Generator view with the corresponding component. The created component is similar to the Anomalies component. It features a sensor selector to choose the sensor or sensor group. Additionally, it displays the current power consumption of the chosen sensor or sensor group. The next addition is an input field to create the artificial anomaly. To insert the newly created artificial anomaly, we create two new bricks to incorporate the Anomaly Generator into our existing flow, consisting of the HIPE machine bricks. When we create a new anomaly, it is sent to the webserver of the Anomaly Generator brick. Afterward, it is merged with the next incoming data record for the corresponding machine sensor or sensor group. Figure 3.6 shows the newly created view for the Anomaly Detection component.
3. Approach

Figure 3.6. Screenshot of the Anomaly Generator view

Generator. The first new brick is called Anomaly Generator. It inherits the InletBrickBase class from the Titan Flow Engine library, which corresponds to the Inlet Brick we described in Section 2.2.1. This brick provides a FastAPI web server to receive the artificial anomalies from the Anomaly Generator view of the Titan Control Center. The second brick is the Anomaly Apply brick which inherits the BrickBase class, and it corresponds to the General brick that we mentioned in Section 2.2.1. Furthermore, the brick has two input ports and one output port. This brick aims to merge the artificial anomalies from the Anomaly Generator with the predicted measurements from the machine sensors and output the modified records to the CCAdapter brick. Additionally, Figure 3.7 shows a simple example for a flow with the added functionality to create artificial anomalies.

3.4 Extend the Visualizations

To extend the visualizations of the Titan Control Center, we are implementing three new graphs. The first one is the graph to display each sensor’s partial power consumption, like the composition chart on the Dashboard. For this graph, we create a new Vue component with a stacked area chart from the charting library Britecharts. To fetch the necessary data, we use the already existing REST API endpoint from the History microservice. Figure 3.8 shows the finished implementation in the Titan Control Center.
3.4. Extend the Visualizations

Figure 3.7. A Flow with the Anomaly Generator and Anomaly Apply Brick

Figure 3.8. Screenshot of Partial Power Consumption Graph of the Titan Control Center

The next extension of the Titan Control Center is implementing two new graphs for the Sensor Detail category. The two new graphs display the monthly and yearly statistics for the selected sensor or sensor group. The Monthly graph displays the average and maximum
3. Approach

Figure 3.9. Screenshot of the Monthly/Yearly Statistics Graphs of the Titan Control Center

power consumption over a month. Additionally, the Yearly Statistics graph provides the user with information for the average and maximum power consumption of the selected sensor or sensor group. Furthermore, the graph displays the data for the last ten years. To collect the necessary data, we extended the Statistics microservice. After that, we use Kafka Streams to create the corresponding topics, one for each graph, to gather and aggregate the needed data. Furthermore, we modified the REST API of the Statistics microservice to serve our new data. Afterwards, we create three new Vue components to house the new graphs. The first component is the StackedAreaChart component which imports the StackedAreaChartMonthly and StackedAreaChartYearly component. Furthermore, the StackedAreaChartMonthly and StackedAreaChartYearly components import and used the stacked bar chart from the Britecharts library and fetched the corresponding data from the REST API endpoint of the Statistic microservice. The Figure 3.9 show the final implementation into the Titan Control Center.
Chapter 4

Usability Evaluation

In this chapter, we evaluate the Titan platform with the added components of the previous chapter. To achieve this, we experiment with one test person to test the whole system’s usability. Additionally, the test person is familiar with the Titan Flow Engine but not with the Titan Control Center. Afterward, we continue by presenting and discussing the evaluation results and considering possible threats to validity.

4.1 Goals

The goal of the conducted evaluation is to test and examine the usability of the developed system. The user group for this kind of platform are operation teams in industrial production environments and researcher so that we can presuppose a general knowledge about the usage of software. As we mentioned before, we use the experiment to test the system’s overall usability rather than compare results.

4.2 Methodology

In the experiment, we evaluate the overall usability of the system. To achieve this, we issue specific tasks that our test person should solve. The tasks are based on the learnings of Holzinger [2005]. The five important usability criteria are as follow:

1. **Learnability** Describes how fast the user can start to work.
2. **Efficiency** Users who spent more time with the system can gain more productivity.
3. **Memorability** Users remember the system and do not need to relearn every functionality.
4. **Low error rate** The provided functionalities lower the susceptibility to errors.
5. **Satisfaction** The system is satisfying to use.

After that, we conduct a interview to gather feedback regarding the usability and possible future enhancements [Ralph et al. 2021]. The experiment and the interview were handled over an online conference tool and the test person used the browser Firefox to open the
4. Usability Evaluation

two web applications. At the start of the experiment, we introduced the Titan Platform, Titan Control Center, and the Titan Flow Engine.

Furthermore, we briefly summarized the production chain of the factory for task T4.5. Then we started with the experiment and each task was communicated orally. The experiment itself had no time limit since we are only interested in the usability of the system. Additionally, we noted down issues and our perceived usability during the experiment and besides that, the test person was allowed to ask questions or request hints at any time. After each task was solved, we conducted the interview with an informal pluralistic walkthrough to identify possible improvement proposals from the test person [Nielsen 1994; Holzinger 2005].

4.3 Experimental Setup

We conduct this experiment to check the system’s overall usability with a specific focus on the graphical user interface. Furthermore, the evaluation tasks are based on the five essential usability characteristics to derive the overall usability from the experiment [Holzinger 2005]. Additionally, we designed the following set of tasks for the test person, which covers all essential features of the system [Henning et al. 2021]. Table 4.1 shows the prepared set of tasks for our test person. The tasks T1, T2, and T3 are supposed to introduce the test person to the system and to check how fast the test person can start his work. These tasks are created to investigate the memorability of the system. On the other hand, T4 with each subtask confronts the test person with a series of tasks to check if the test person can obtain new knowledge and then use the obtained knowledge in repetitive/similar tasks. Additionally, these tasks cover the efficiency and learnability criteria. Afterward, the combination of T3 and T5 examines if the user can apply the obtained knowledge to solve an unknown but similar task. This covers the memorability criteria. Following in the next two paragraphs, we describe the used hardware and software components for the usability evaluation.

Hardware  The demo environment is hosted on a server and the specification are presented in Table 4.2. The test person used a browser on his own hardware to open the two frontend applications from the demo environment.

Software  The Titan Platform is deployed with a Docker-Compose file and each service runs in a separate Docker container. Additionally, the concrete versions are listed in Table 4.3.
4.3. Experimental Setup

Table 4.1. Tasks for the test person

<table>
<thead>
<tr>
<th>Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
</tr>
<tr>
<td>T2</td>
</tr>
<tr>
<td>T3</td>
</tr>
<tr>
<td>T4</td>
</tr>
<tr>
<td>T4.1</td>
</tr>
<tr>
<td>T4.2</td>
</tr>
<tr>
<td>T4.3</td>
</tr>
<tr>
<td>T4.4</td>
</tr>
<tr>
<td>T4.5</td>
</tr>
<tr>
<td>T4.6</td>
</tr>
<tr>
<td>T5</td>
</tr>
<tr>
<td>T6</td>
</tr>
<tr>
<td>T7</td>
</tr>
</tbody>
</table>

Table 4.2. Hardware specifications from the server for the usability evaluation

<table>
<thead>
<tr>
<th>OS</th>
<th>Ubuntu 20.04.2 LTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kernel</td>
<td>5.4.0-67-generic</td>
</tr>
<tr>
<td>CPU</td>
<td>Intel Xeon E5-2630v4 @2.20GHz</td>
</tr>
<tr>
<td>RAM</td>
<td>30GB</td>
</tr>
</tbody>
</table>

Table 4.3. Software specification for the usability evaluation

<table>
<thead>
<tr>
<th>Docker</th>
<th>Version 19.03.08</th>
</tr>
</thead>
<tbody>
<tr>
<td>Docker-Compose</td>
<td>Version 1.25.0</td>
</tr>
</tbody>
</table>
4. Usability Evaluation

4.4 Results

In this section, we present the results of the usability experiment and the interview.

**T1** The test person used the browser to open the Titan Control Center Frontend and found the Titan Flow Engine link after approximately 10 seconds. He did not ask any questions or requested hints.

**T2** The test person used the provides user information to log into the prepared demo account of the Titan Flow Engine. Quickly after that, he found the page to upload the bricks. He did not ask any questions or requested hints.

**T3** The test person quickly found the specific frontend page to create and start a new flow. He did not ask any questions or requested hints.

**T4** The test person quickly solved this task without asking questions or requesting hints.

**T4.1** In the beginning, the test person was uncertain which graph to use and scrolled through the frontend, but he settled with the Contribution Chart under the Sensor Details. He did not ask any questions or requested hints.

**T4.2** The test person used the Weekly/Daily Course graph to solve the task. He did not ask any questions or requested hints.

**T4.3** The test person quickly found the indicator on the dashboard and solved the given task without asking questions or requesting hints.

**T4.4** The test person quickly used the Comparison tool to compare the two sensors, but he struggled to select the right time on the time picker tool. He did not ask any questions or requested hints.

**T4.5** The test person used the comparison tool, selected the two sensors, and asked a question about the production chain. After that, he solved the task.

**T4.6** The goal was to find the predicted power consumption for the Chip Press sensor in 30min. The test person tried to find a solution under the Sensor Details category for the given sensor. Afterward, he clicked through the categories and used the Forecast category. He did not ask any questions or requested hints.

**T5** This task was solve rapidly by the test person and he did not ask any questions or requested hints.
4.5 Discussion

**T6** The test person’s first choice was the Anomalies category, but then he switched to the Anomaly Generator category. He did not ask any questions or requested hints.

**T7** The test person used the Anomalies category and selected the sensor to identify the anomaly successfully. He did not ask any questions or requested hints.

**Interview** At the start of the interview, we asked the test person about his satisfaction while using the system to solve the given tasks. The test person pointed out that the overall look of the web applications is very modern and pleasing. Furthermore, he stated that he liked the matching color theme of the Titan Control Center graphs. Additionally, he pointed out that the Titan Control Center is has a slim layout and the user is not overwhelmed with the number of presented graphs and information.

After that, we continued with the informal pluralistic walkthrough and asked the test person to identify possible graphs and overall layout improvements. He mentioned that he was unsure what type of information the Weekly Consumption and Daily Composition graphs provided and suggested a legend to explain the presented information further. Additionally, he suggested moving or replacing the Partial Power Consumption graph because it provides nearly the same information as the Composition chart.

4.5 Discussion

In this section, we discuss the presented results of the previous section.

**T1** This task was designed to introduce the test person to the system, the art of tasks for the evaluation, and show how fast he can start to work. The test person solved this task without any issues.

**T2/T3** These tasks are designed to test the user’s memorability because he used the system before this evaluation. This is also the reason that the test person quickly solved the given tasks. This points out that the Titan Flow Engine user interface provides a certain memorability.

**T4** This task was quickly solved by the test person and designed to prepare for the subtasks.

**T4.1** In the beginning, the test person was quite unsure which graph to use and traversed multiple times through the web application, but he settled with the Contribution Chart under the Sensor Details. We think that the main reason for the test persons uncertainty is that he never used the Titan Control Center before. If the test person were given more time to explore the Titan Control Center, the result may change because he would have a better
4. Usability Evaluation

overview of the components and functionalities.

**T4.2** This task was designed to test the efficiency of the test person by solving similar tasks. He quickly used the Weekly/Daily Course graph to solve the task.

**T4.3** Following the style of the previous task, this task is also designed to test the user’s efficiency. The test person quickly found the indicator on the dashboard and solved the given task.

**T4.4** In addition to the last two tasks, this task also tests the efficiency of the user. The test person quickly found the right Comparison tool to compare the two sensors, but he struggled to select the right time on the time picker tool. He later pointed out that he thought that the second calendar was for the end date and not a preview for the next month.

**T4.5** This task was also solved rapidly by the test person after asking a domain-specific question about the production process. After that, he successfully used the Comparison Tool and selected the two sensors and solved the task.

**T4.6** This task also examines the efficiency of the user. The test person tried to find a possible solution under the Sensor Details category for the given sensor on the first try. Afterward, he clicked through the categories and used the Forecast category. After the task, he mentioned that he found the category name ”Forecast” misleadingly.

**T5** Task 5 combined with Task 3 was designed to test the user’s memorability by giving the user an unknown problem. The user had to transfer the learning of T3 to T5 to solve the task. In the end, the test person solved the problem rapidly.

**T6** The following task was also designed to test the user’s efficiency by providing a similar task like T3. The test persons first choice was the Anomalies category, but then he switched to the Anomaly Generator category. He later mentioned that the Anomaly category was over the Anomaly Generator category and because of that he clicked on this category before choosing the correct one.

**T7** The last task was designed to cover the last feature of the Titan Control Center and the test person successfully selected the sensor to identify the anomaly.

**Interview** The interview revealed that the test person enjoyed working with the system. The overall design is modern and the user is not overwhelmed with the number of presented graphs and information. Additionally, the user criticized the lack of additional information on the Weekly Consumption and Daily Composition heatmaps. The test person proposed a legend to better categorize the power consumption compared to the other fields.
on the heat map or a tooltip if you hover over the fields of the heatmap. Furthermore, he suggested moving or replacing the Partial Power Consumption graph because it provides nearly the same information as the Composition chart above. The user will most likely take the Composition Chart to get the needed information because it is located over the Partial Power Consumption chart.

4.6 Threats to Validity

Our conducted usability experiment with only one test person does not provide a sufficient foundation for reasonable and comparable results. Nonetheless, as we mentioned before, the usability experiment’s goal is to collect indications for the overall usability of the system rather than comparing the results to other systems. To collect sufficient data for an evaluation, an experiment with a significantly increased number of test persons should be conducted [Nielsen and Landauer 1993]. Furthermore, the experiment should cover different skilled people to collect more meaningful data since our test person already worked with the Titan Flow Engine the results may vary for other test persons.
The Titan Control Center’s showcase is based on the prototype introduced by Henning [2018]. Within the Titan platform, the Titan Control Center is a scalable architecture for a system that offers components to collect, display, and analyze power consumption data from various sources (Section 2.2.2). Our showcase also uses the Time Series Plot by Koch [2020] for the main graph to display the power consumption. Furthermore, the Forecast service embedded in the showcase is provided by Boguhn [2020]. It creates a forecast for every sensor in the showcase and stores the forecasts in an OpenTSDB. Besides that, the Forecasts are accessible via a REST API.

To complete the Titan Control Center’s showcase, we deployed the Titan Flow Engine to simulate the machine sensors of a production environment to create realistic data. For this, we created new bricks for the Titan Flow Engine based on the machines of the dataset from Bischof et al. [2018] and composited a new flow. The mentioned dataset contains measurements from the electronics production site from the Institute of Data Processing and Electronics at the Karlsruhe Institute of Technology in Germany.

The collected and processed data from monitoring and analyzing the industrial environment using the Titan Control Center is a viable prototype to achieve the Industrial DevOps approach described by Hasselbring et al. [2019]. The authors of the paper introduced a concept of a new cyclic, continuous adaptation and improvement process to the existing production environment before releasing to production.
Chapter 6

Conclusion and Future Work

6.1 Conclusion

In the first chapter, we defined five goals for our thesis. The thesis’s first goal was to develop and implement the data generator based on the HIPE data set. We cleaned and prepared the HIPE data set in the first step and used this data to train a linear regression-based model for each machine sensor of the HIPE data set. After that, we needed to scale some trained models’ predictions to match the actual readings’ values better. Afterward, we used the Titan Flow Engine to create bricks for each machine sensor with our trained model and build a flow with created bricks.

The second goal of our thesis was to create a new user interface component for the Forecast microservice. This allows the user to analyze the predicted power consumption of the industrial production environment to detect possible spikes. We created the corresponding Vue components to provide the needed functionality and created a new category called Forecast.

The next goal was to extend the current visualization capabilities of the Anomaly Detection microservice, which allows the user to detect anomalies in the monitored system quickly. Additionally, the user can filter the anomalies based on their anomaly score. This thesis extended the Dashboard with a table that displays all detected anomalies based on the selected anomaly score. Furthermore, we created a new category called Anomaly Generator with the corresponding Vue components. The new component allows the user to create artificial anomalies, and the newly created anomalies are send to the corresponding brick in our flow.

The fourth goal was to create new visualizations and graphs for the Titan Control Center. The first visualization was a new stacked area chart to display the partial power consumption of each sensor. Additionally, we created two new bar charts to display monthly and yearly power consumption statistics for each sensor.

The last goal was the evaluation of the Titan Control Center with a test person. The conducted experiment to analyze the system’s usability shows that the user interface provides a simple to use and powerful tool to analyze the incoming data. This result is based on the task-based experiment and the qualitative survey at the end with the test person. The test person solved all tasks and afterward, he provided helpful feedback to improve the graphical user interface. In summary, the evaluation reveals that the Titan
6. Conclusion and Future Work

Platform, and especially the Titan Control Center, provides a graphical user interface with high usability.

We provide our complete implementation as a package [Grabitzky 2021] that contains the docker-compose file to start the project and the source code of each microservice that we modified. Furthermore, we provide the code for the bricks as well as the finished zip file.

6.2 Future Work

While working on this thesis, we identified multiple aspects that could be considered for future work. The first identified future work is for the data generator based on the HIPE data set. We used a simple linear regression-based approach to train the models. Another approach could be to use a more sophisticated model for each machine sensor to generated a more precise prediction. Another step could be to use a more extensive dataset that features more seasonal dependencies than the HIPE data set because the HIPE data set covers only three months’ worth of data.

The next potential future work is to further improve the Titan Control Center’s visualizations based on the provided feedback by this thesis. Moreover, we pointed out in Section 4.6 that the number of test persons for the evaluation was small, and a new experiment with a significantly increased number of test persons should be conducted. The new evaluation would provide more representative data for a sound evaluation of the usability of the system.


