

Analytics As A Service

Table of Contents

1. Aufwand.....	2
2. Pakete	2
3. Aufbau	2
4. Übersicht.....	2
5. Authentifizierung	4
6. Interceptor für Ladeanzeige und Fehlerbehandlung.....	6
7. Metriken.....	9
8. Detektoren	13
9. Logs	16
10. Clients.....	17

1. Aufwand

Der Aufwand betrug aufgrund Vorerfahrung ~15h.

2. Pakete

Verwendet wurde folgende Pakete:

- Angular: `@angular 13.1.1`
- Authentifizierung: `angular-oauth2-oidc 13.0.1`
- Styling: `@angular/material 13.1.1`
- Charts: `ng2-charts 3.0.5`

3. Aufbau

Das Projekt wurde strukturell auf folgende Teile aufgeteilt:

- components (Angular Komponenten)
- guards (AuthGuard)
- interceptors (HTTP-Request Interceptor)
- models (Interfaces und Enums)
- pipes (Pipes zum Darstellen von Enums)
- services (Services zum Zugreifen auf die Web-API, LocalStorage und Anzeige von Ladeanimationen)
- utils (Klassen mit statischen Hilfsmethoden wie zum Beispiel für die Formatierung von Validierungsfehlern)

4. Übersicht

Ganz oben wird eine Navbar angezeigt, wo man sich entsprechend ab-/anmelden kann. Darunter wird eine Angular-Sidebar-Komponente angezeigt, wo die entsprechenden Pages ausgewählt werden können. Dies ist aber nur möglich, wenn man auch angemeldet ist.

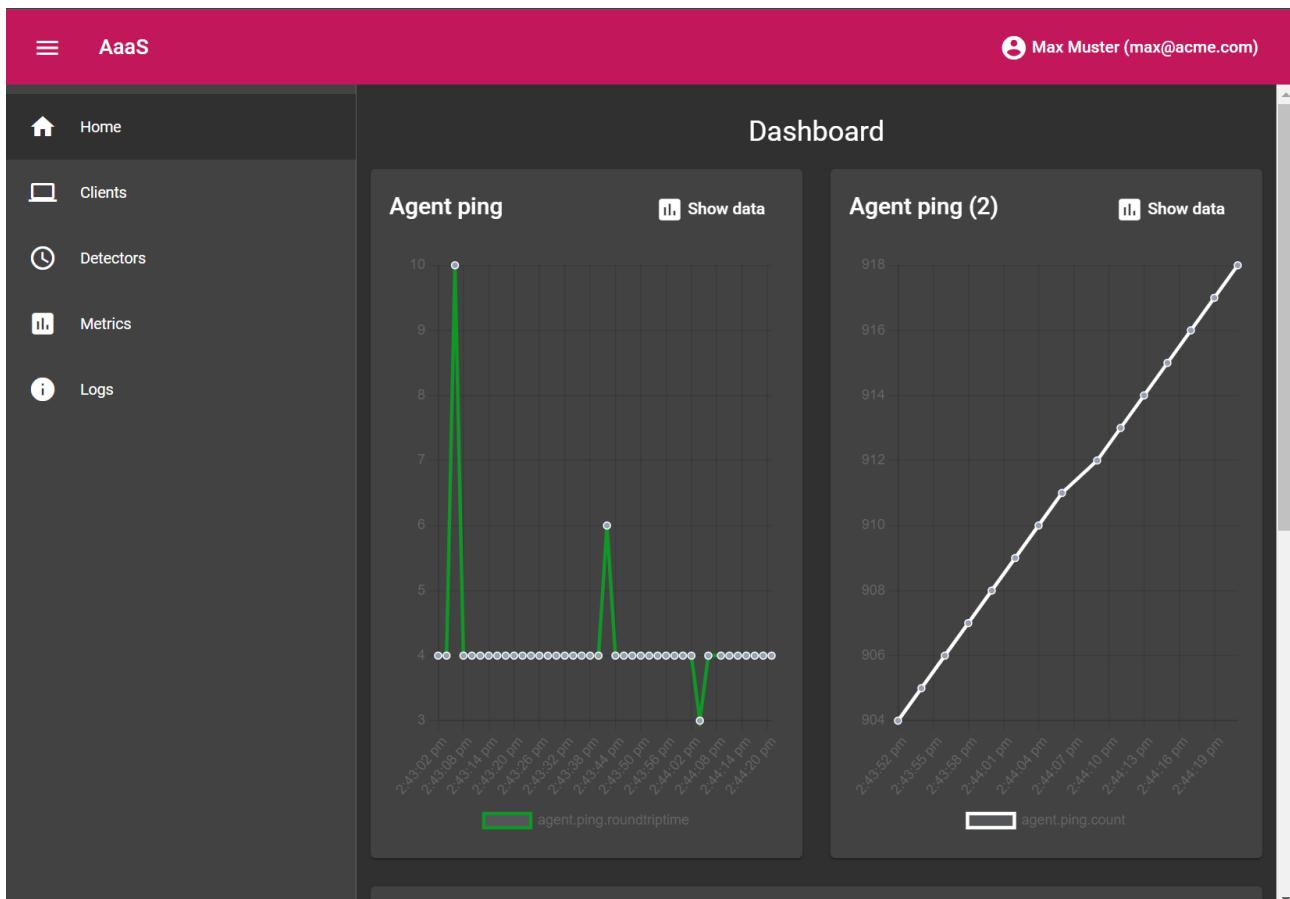


Figure 1. Übersicht

5. Authentifizierung

Hier wurde kein eigener Keycloak-Server verwendet, anstelle einfach der Identity-Server von Manfred Steyr.

Listing 1. auth.config.ts

```
import { AuthConfig } from 'angular-oauth2-oidc';

export const authConfig: AuthConfig = {

  // Url of the Identity Provider
  issuer: 'https://steyer-identity-server.azurewebsites.net/identity',

  // URL of the SPA to redirect the user to after login
  redirectUri: window.location.origin + '/index.html',

  // The SPA's id. The SPA is registerd with this id at the auth-server
  clientId: 'spa-demo',

  // set the scope for the permissions the client should request
  // The first three are defined by OIDC. The 4th is a usecase-specific one
  scope: 'openid profile email voucher',
}
```

Es wurde ein eigener Service verwendet, der entsprechend auf die Methoden vom OAuthService zugreift.

Listing 2. *auth.service.ts*

```
import { Injectable } from '@angular/core';
import { OAuthService } from 'angular-oauth2-oidc';
import { JwksValidationHandler } from 'angular-oauth2-oidc-jwks';
import { authConfig } from '../auth.config';

@Injectable({
  providedIn: 'root'
})
export class AuthService {

  constructor(private oauthService: OAuthService) { }

  signIn(): boolean {
    this.oauthService.initImplicitFlow();
    return true;
  }

  configure() {
    this.oauthService.configure(authConfig);
    this.oauthService.tokenValidationHandler = new JwksValidationHandler();
    this.oauthService.loadDiscoveryDocumentAndTryLogin();
  }

  getClaims() {
    return this.oauthService.getIdentityClaims();
  }

  signOut(): boolean {
    this.oauthService.logout(true);
    return true;
  }

  isSignedIn() {
    return this.oauthService.hasValidAccessToken() &&
      this.oauthService.hasValidIdToken();
  }
}
```

Mittels Claims werden Daten wie Name und Email ermittelt und angezeigt.

6. Interceptor für Ladeanzeige und Fehlerbehandlung

Mittels eigenem Interceptor, welcher zwischen jeder HTTP-Anfrage ausgeführt wird werden Ladenanimationen angezeigt und es erfolgt die Fehleranzeige.

Listing 3. api-interceptor.ts

```
import { HttpEvent, HttpHandler, HttpInterceptor, HttpRequest, HttpResponse } from
"@angular/common/http";
import { Injectable } from "@angular/core";
import { Observable, throwError } from "rxjs";
import { catchError, map } from "rxjs/operators";
import { LoaderService } from "../services/loader.service";
import { MessageService } from "../services/message.service";

@Injectable()
export class ApiInterceptor implements HttpInterceptor {

  constructor(
    private loaderService: LoaderService,
    private messageService: MessageService) { }

  intercept(request: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {

    this.loaderService.setLoading(true, request.url);
    return next.handle(request)
      .pipe(catchError((error) => {
        this.messageService.showErrorMessage(error);
        this.loaderService.setLoading(false, request.url);
        return throwError(error);
      })))
      .pipe(map<unknown, any>(event => {
        if (event instanceof HttpResponse) {
          this.loaderService.setLoading(false, request.url);
        }
        return event;
      })));
  }
}
```

Der Message-Service, welcher für die Anzeige von Fehlern verwendet wird, verwendet intern eine Angular Material Komponente namens Snackbar. Hier gibt es eine Methode zum Anzeigen von Infos und und Fehlern.

Listing 4. message.service.ts

```
import { Injectable } from '@angular/core';
import { MatSnackBar } from '@angular/material/snack-bar';

@Injectable({
  providedIn: 'root'
})
export class MessageService {

  private static duration = 6000;

  constructor(private snackBar: MatSnackBar) { }

  showErrorMessage(error: string | {status: number}) {

    console.error(error);

    let message: string;

    if (typeof(error) === 'string') {
      message = error;
    } else if (error.status == 0) {
      message = "Can't connect to server.";
    } else if (error.status == 404) {
      message = "Entity does not exist."
    } else {
      message = `Error performing request, status code = ${error.status}`;
    }

    return this.showMessage(message, "message-error");
  }

  showInfoMessage(message: string) {
    return this.showMessage(message, "message-info");
  }

  private showMessage(message: string, cssClass: string) {
    return this.snackBar.open(message, undefined,
      { duration: MessageService.duration, panelClass: cssClass })
  }
}
```

Zum Anzeigen einer Ladeanimation wird eine LoaderService mit einer entsprechenden Komponente verwendet, welche einen Loading-Indicator anzeigt, wenn aktuell eine Http-Anfrage durchläuft. Für Autovervollständigung kann diese Ladeanimation dann vorübergehend deaktiviert werden.

Listing 5. loader.service.ts

```
import { Injectable } from '@angular/core';
import { BehaviorSubject } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class LoaderService {
  loadingSub: BehaviorSubject<boolean> = new BehaviorSubject<boolean>(false);

  loadingMap: Map<string, boolean> = new Map<string, boolean>();

  disabled: boolean = false;

  constructor() { }

  setLoading(loading: boolean, url: string): void {
    if (!url) {
      throw new Error('The request URL must be provided to the LoaderService.setLoading function');
    }
    if (loading === true) {
      this.loadingMap.set(url, loading);
      this.loadingSub.next(true);
    } else if (loading === false && this.loadingMap.has(url)) {
      this.loadingMap.delete(url);
    }
    if (this.loadingMap.size === 0) {
      this.loadingSub.next(false);
    }
  }

  disableLoading() {
    this.disabled = true;
  }

  enableLoading() {
    this.disabled = false;
  }
}
```


7. Metriken

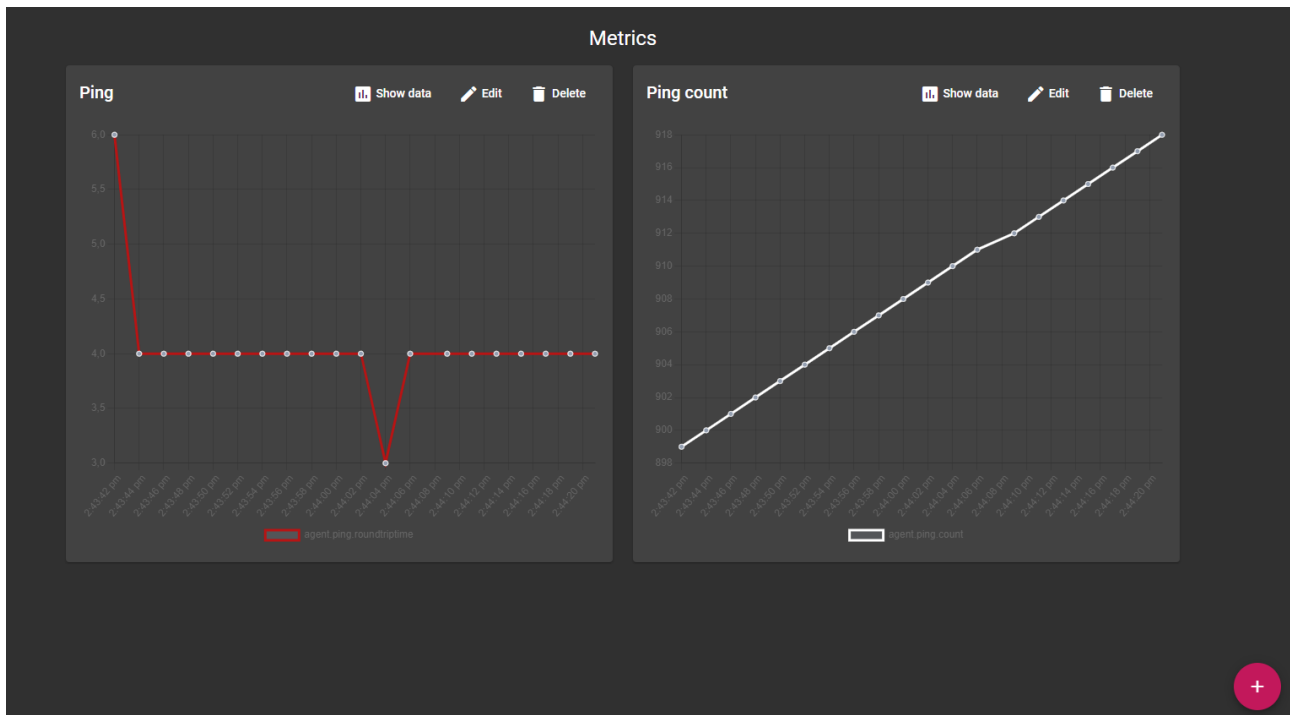


Figure 2. Anzeige von Metriken

Mittels Floating-Action-Button (FAB) rechts unten können neue Metriken hinzugefügt werden. Daraufhin wird ein Angular-Material Dialog geöffnet, wo entsprechende Einstellungen gesetzt werden können.

Figure 3. Hinzufügen von Charts

Hier wird dann eine Reactive-Form angezeigt, wo eine entsprechende Konfiguration gesetzt werden kann. Pflichtfelder werden mit einem * markiert und bei Fehlern wird auch entsprechend eine Fehlermeldung angezeigt und es kann nicht mehr gespeichert werden. Felder wie die Metriknamen wurden dabei mit Material Autocompletes umgesetzt und folgendermaßen konfiguriert:

Listing 6. edit-metric-chart-dialog.ts

```
this.filteredInstanceIds = this.form.get('instanceId')?.valueChanges.pipe(  
  debounceTime(300),  
  distinctUntilChanged(),  
  tap(() => this.loaderService.disableLoading()),  
  switchMap(term => this.instanceService.search(term)),  
  tap(() => this.loaderService.enableLoading())  
);
```

Figure 4. Hinzufügen von Charts (ausgefüllt)

Die Metriken werden nach dem Speichern in einer Übersicht angezeigt. Das Speichern selbst erfolgt im LocalStorage, dazu wurde ein eigener Service erstellt.

Listing 7. StorageService.ts

```
import { Injectable } from '@angular/core';

@Injectable({
  providedIn: 'root'
})
export class StorageService {

  static readonly metricConfiguration: string = "METRIC_CHART_CONFIGURATION";

  constructor() {}

  set<T>(key: string, value: T) {
    localStorage.setItem(key, JSON.stringify(value));
  }

  get<T>(key: string, fallbackValue: T): T {
    const item = localStorage.getItem(key);

    if(item){
      return JSON.parse(item);
    } else {
      return fallbackValue;
    }
  }
}
```

Nach dem Hinzufügen werden alle Metrik-Charts in einer Übersicht angezeigt.

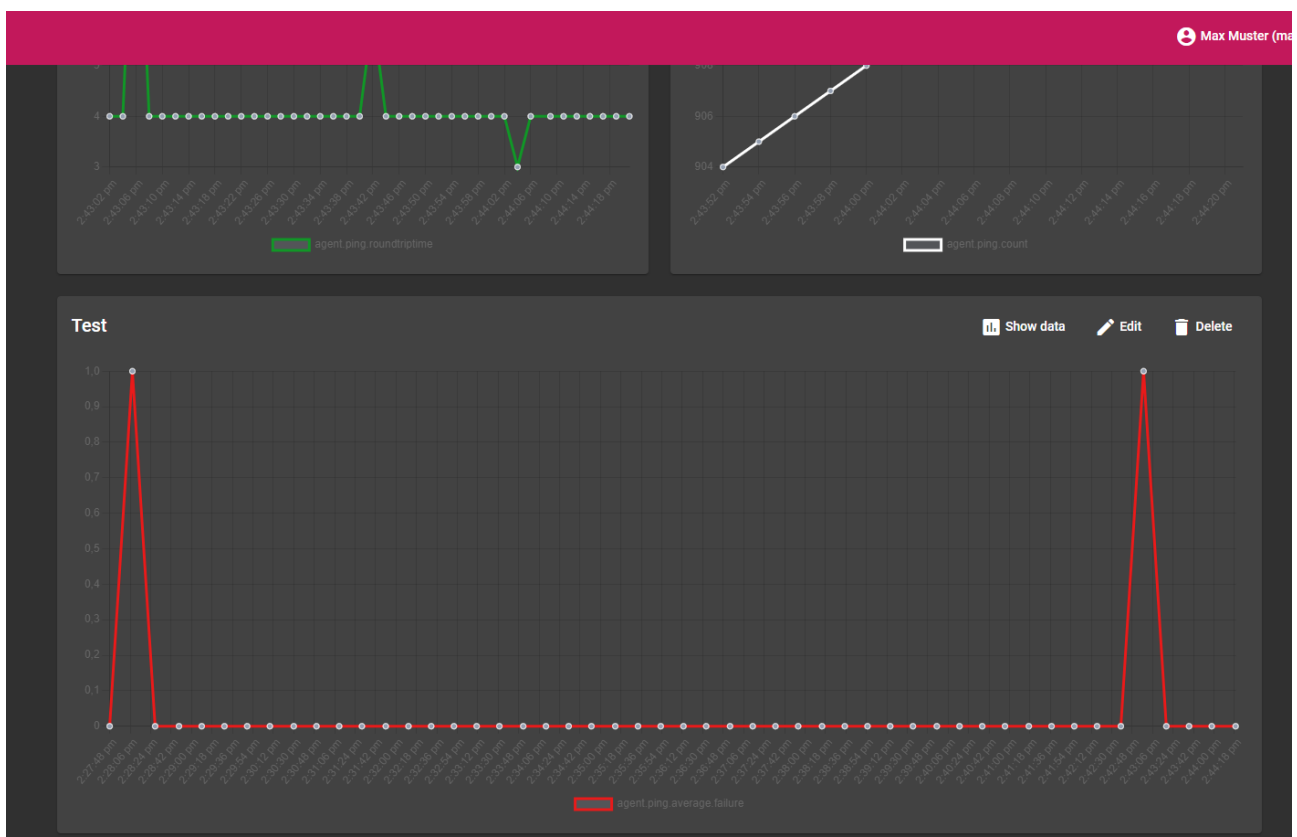
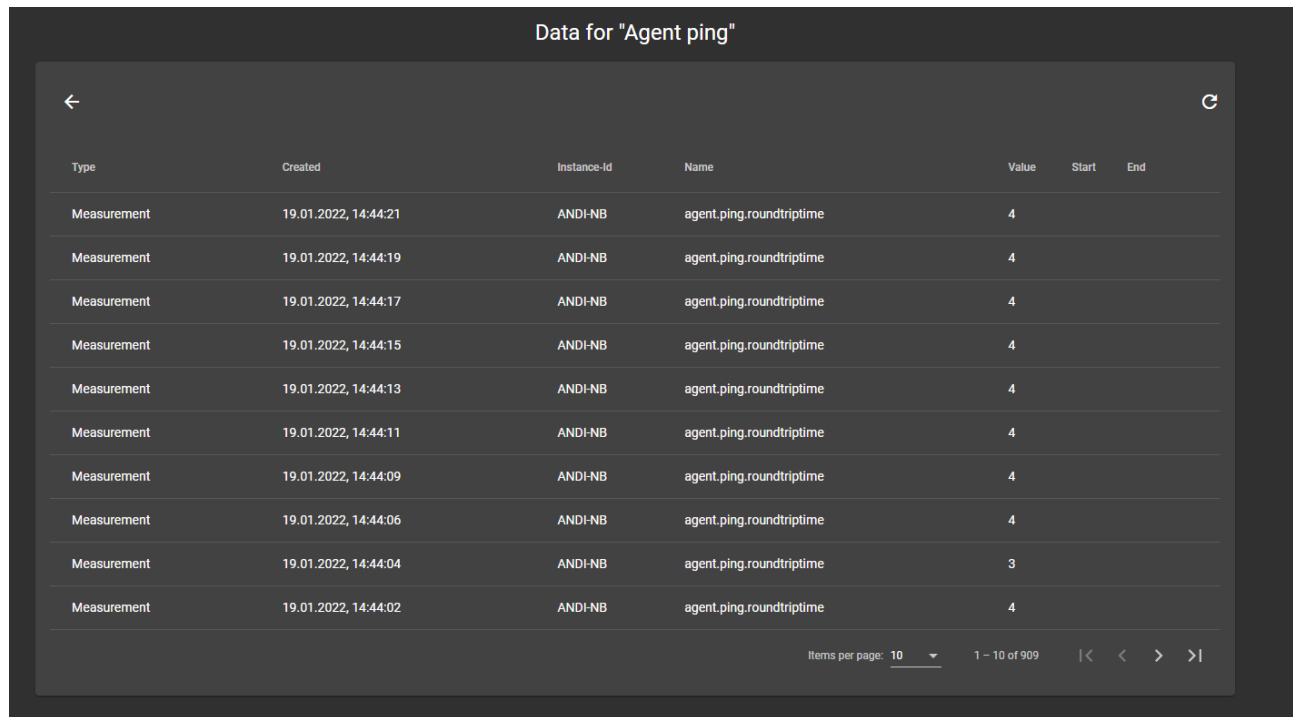


Figure 5. Anzeige der hinzugefügten Metrik

Mittels "Edit" und "Delete" können diese auch nach der Erstellung noch bearbeitet bzw. gelöscht werden. Mit "Show data" werden dazugehörige Daten eines Charts angezeigt.



Data for "Agent ping"

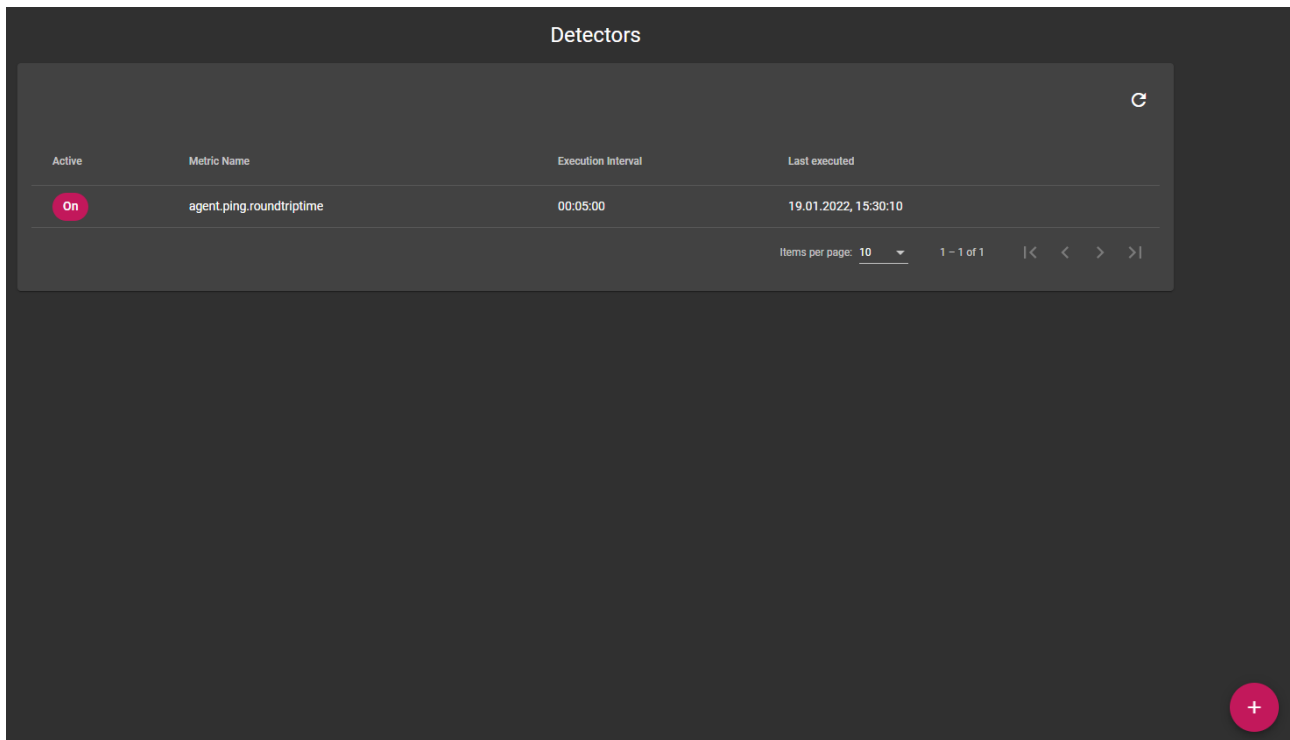
Type	Created	Instance-Id	Name	Value	Start	End
Measurement	19.01.2022, 14:44:21	ANDI-NB	agent.ping.roundtripTime	4		
Measurement	19.01.2022, 14:44:19	ANDI-NB	agent.ping.roundtripTime	4		
Measurement	19.01.2022, 14:44:17	ANDI-NB	agent.ping.roundtripTime	4		
Measurement	19.01.2022, 14:44:15	ANDI-NB	agent.ping.roundtripTime	4		
Measurement	19.01.2022, 14:44:13	ANDI-NB	agent.ping.roundtripTime	4		
Measurement	19.01.2022, 14:44:11	ANDI-NB	agent.ping.roundtripTime	4		
Measurement	19.01.2022, 14:44:09	ANDI-NB	agent.ping.roundtripTime	4		
Measurement	19.01.2022, 14:44:06	ANDI-NB	agent.ping.roundtripTime	4		
Measurement	19.01.2022, 14:44:04	ANDI-NB	agent.ping.roundtripTime	3		
Measurement	19.01.2022, 14:44:02	ANDI-NB	agent.ping.roundtripTime	4		

Items per page: 10 1 – 10 of 909

Figure 6. Detailansicht eines Metrik-Charts

8. Detektoren

In einer Tabelle werden Detektoren dargestellt. Wenn man auf den Button bei der Spalte active drückt, wird der aktuelle Detektor aktiviert/deaktiviert.



The screenshot shows a web interface titled "Detectors". It features a table with the following columns: "Active", "Metric Name", "Execution Interval", and "Last executed". There is a single row in the table with the following data: "On" (with a red circle icon), "agent.ping.roundtripTime", "00:05:00", and "19.01.2022, 15:30:10". Below the table, there is a pagination bar showing "Items per page: 10", "1 - 1 of 1", and navigation arrows. On the right side of the interface, there is a vertical sidebar with a red circle icon containing a white plus sign at the bottom.

Active	Metric Name	Execution Interval	Last executed
On	agent.ping.roundtripTime	00:05:00	19.01.2022, 15:30:10

Figure 7. Übersicht von Detektoren

Detaillierte Informationen werden angezeigt, wenn man den Detektor auswählt.

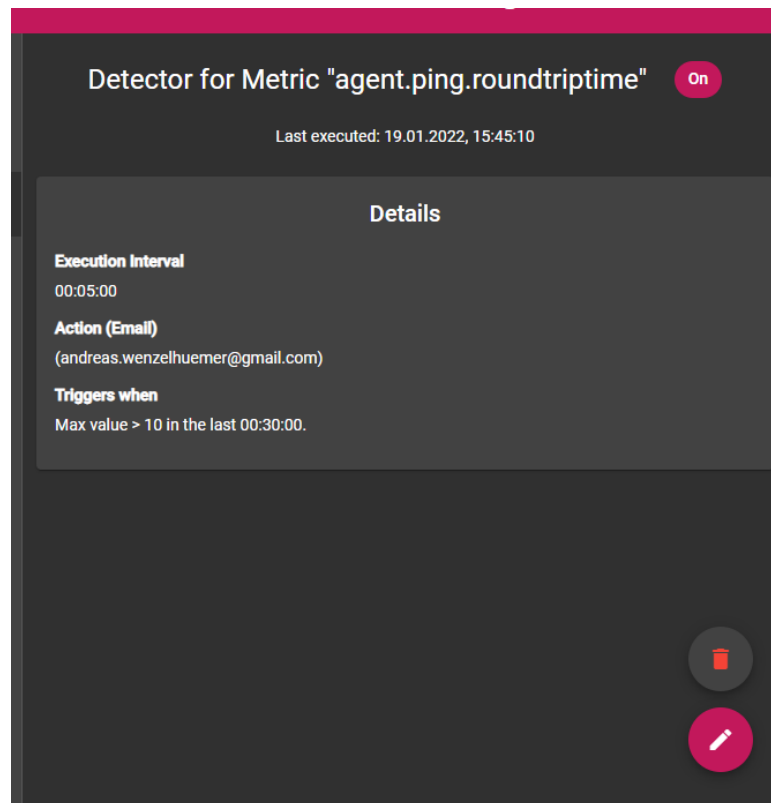


Figure 8. Detailansicht eines Detektors

Detektoren können auch gelöscht oder verändert werden. Dazu werden zwei Floating-Action-Buttons zum Bearbeiten und Löschen angezeigt.

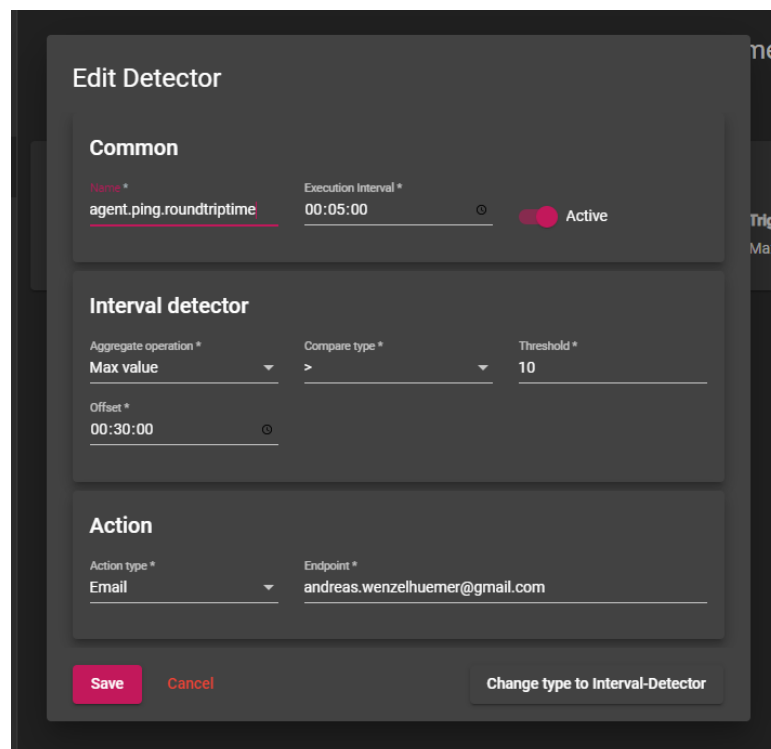


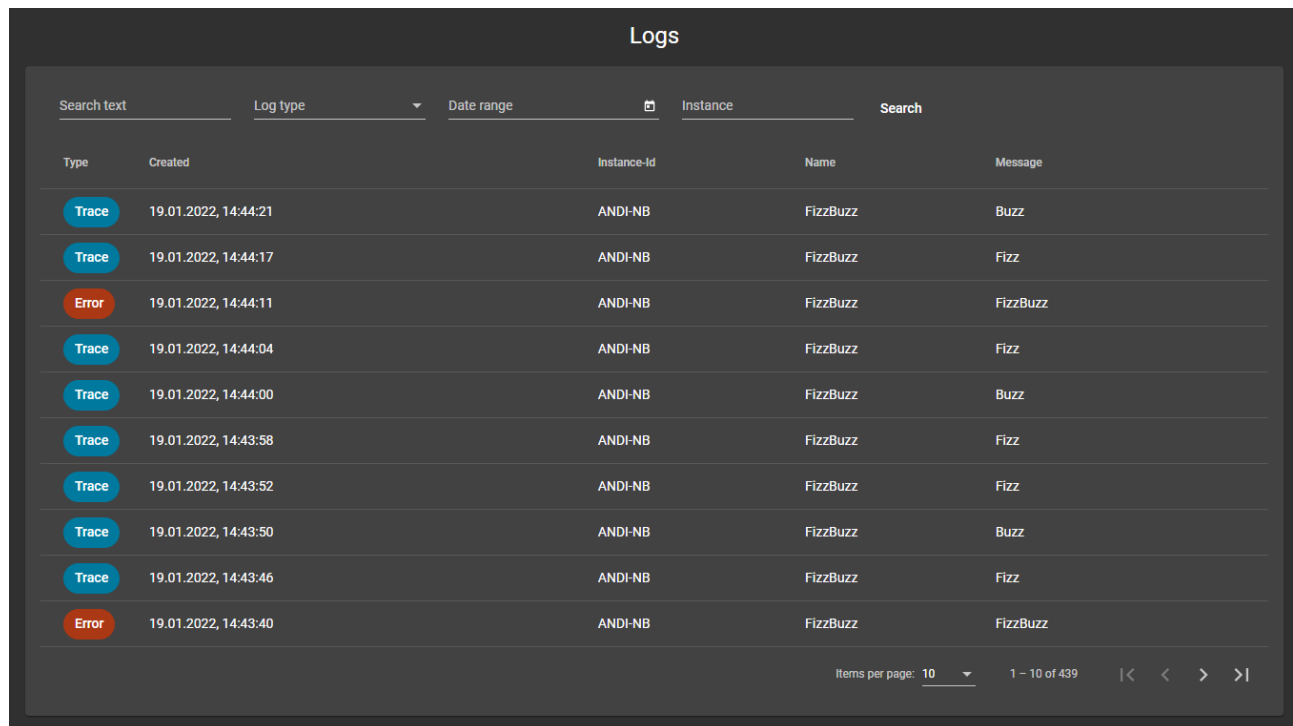
Figure 9. Bearbeiten von Detektoren

Auch hier wurde wieder mit Reactive-Forms gearbeitet. Es können auch zwischen den

zwei vorhandenen Detektortypen hin und her gewechselt werden, bestehende Daten werden dabei gelöscht.

9. Logs

Logs werden in einer Übersicht dargestellt und können entsprechend gefiltert werden. Die einzelnen Log-Levels werden in unterschiedlichen Farben angezeigt. Für die Filterung wurde eine eigene Komponente erstellt, wo wieder Reactive-Forms zum Einsatz gekommen ist. Mithilfe eines Output-Emitters werden bei Änderungen die Filter an die Übersicht weitergegeben.



The screenshot shows a 'Logs' interface with a dark theme. At the top, there are filter controls: 'Search text', 'Log type' (a dropdown menu), 'Date range' (with a calendar icon), 'Instance' (with a dropdown arrow), and a 'Search' button. Below these is a table with the following columns: 'Type', 'Created', 'Instance-Id', 'Name', and 'Message'. The table contains 10 log entries. The 'Type' column uses colored circles to indicate the log level: blue for 'Trace' and red for 'Error'. The 'Created' column shows timestamps from 19.01.2022, 14:43:40 to 19.01.2022, 14:44:21. The 'Instance-Id' column shows 'ANDI-NB' for all entries. The 'Name' column shows 'FizzBuzz' for all entries. The 'Message' column shows 'Buzz' or 'Fizz'. At the bottom right, there is a pagination control showing 'Items per page: 10' and '1 - 10 of 439', along with navigation arrows.

Type	Created	Instance-Id	Name	Message
Trace	19.01.2022, 14:44:21	ANDI-NB	FizzBuzz	Buzz
Trace	19.01.2022, 14:44:17	ANDI-NB	FizzBuzz	Fizz
Error	19.01.2022, 14:44:11	ANDI-NB	FizzBuzz	FizzBuzz
Trace	19.01.2022, 14:44:04	ANDI-NB	FizzBuzz	Fizz
Trace	19.01.2022, 14:44:00	ANDI-NB	FizzBuzz	Buzz
Trace	19.01.2022, 14:43:58	ANDI-NB	FizzBuzz	Fizz
Trace	19.01.2022, 14:43:52	ANDI-NB	FizzBuzz	Fizz
Trace	19.01.2022, 14:43:50	ANDI-NB	FizzBuzz	Buzz
Trace	19.01.2022, 14:43:46	ANDI-NB	FizzBuzz	Fizz
Error	19.01.2022, 14:43:40	ANDI-NB	FizzBuzz	FizzBuzz

Figure 10. Anzeige von Logs

10. Clients

Für die Clients gibt es auch eine Übersicht, wobei hier der App-Key verschlüsselt angezeigt wird. Es können aber neue Clients hinzugefügt werden und der App-Key kopiert werden. Dies erfolgt wieder über den Floating-Action-Button rechts unten.

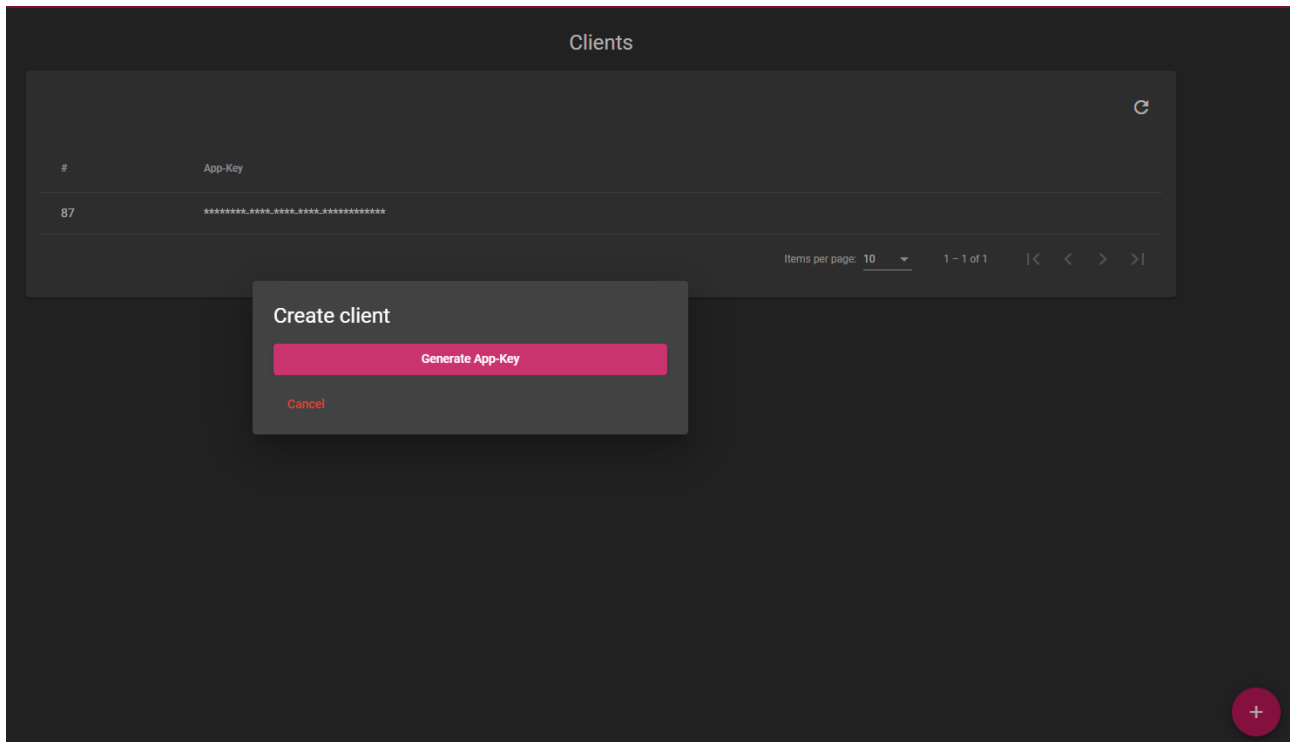


Figure 11. Erstellen neuer Clients

Nach dem Drücken auf "Generate App-Key" wird ein neuer Client erstellt, der App-Key kann danach kopiert werden.

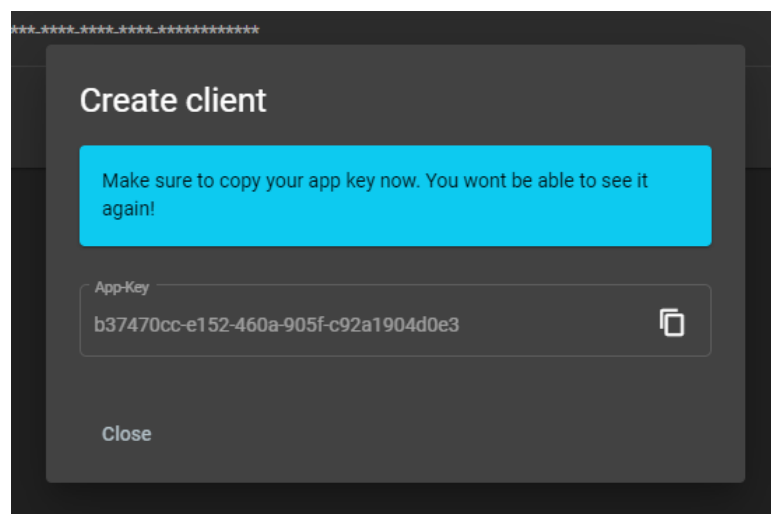


Figure 12. Neuer Client erstellt