IsoPrüfi

Documentation

Table of contents

1. W	elcome to IsoPrüfi	4
1.1	Quick Overview	4
1.2	Getting Started	4
1.3	Architecture	4
1.4	Documentation Structure	5
2. Ge	etting Started	6
2.1	Contribute & Build	6
2.2	Contributing Guide	8
3. De	evelopment	12
3.1	Guidelines / Conventions	12
3.2	Testing Strategy	15
	API Reference	16
3.4	API V1	18
3.5	Troubleshooting Guide	19
4. Co	ode Documentation	25
4.1	Assembly Rest-API	25
4.2	Assembly MQTT-Receiver-Worker	27
4.3	Assembly MQTT-Sender	28
4.4	Assembly Database	29
4.5	Assembly UnitTests	31
4.6	Assembly IntegrationTests	32
4.7	Assembly LoadTests	34
4.8	Contents pages	36
4.9	Index pages	36
4.10	0 Frontend	37
5. Do	ocker	170
5.1	Environment Files	174
6. Sy	stem Documentation	177
6.1	Database Schema	177
6.2	MQTT Protocol Documentation	180
6.3	Monitoring & Observability	182
7. Iso	oPrüfi Documentation	184
7.1	Introduction and Goals	184
7.2	Quality Requirements	186
7.3	Architecture Constraints and Solution Strategy	188

7.0 KISKS allU TECHNICAL DEBIS	201
7.6 Risks and Technical Debts	201
7.5 Context and Scope	199
7.4 Architecture Decisions	191

1. Welcome to IsoPrüfi

We are happy that you are here



IsoPrüfi is an IoT-based system for testing building insulation effectiveness by monitoring temperature differences between indoor and outdoor environments.

1.1 Quick Overview

The system uses Arduino-based sensors to collect temperature data from multiple locations, processes it through a containerized backend, and presents insights via a web interface.

Key Features:

- Real-time temperature monitoring with Arduino sensors
- Automated data collection and analysis
- Web-based dashboard for visualization
- Containerized architecture for easy deployment
- Offline data buffering with SD card storage

1.2 Getting Started

- Setup & Build Initial project setup and development environment
- Docker Environment Container overview and local development
- API Reference REST API documentation
- Guidelines Development conventions and best practices

1.3 Architecture

Built with modern technologies:

- Backend: .NET REST API, MQTT message broker
- Frontend: React with TypeScript
- Database: PostgreSQL + InfluxDB for time-series data

- Hardware: Arduino MKR WiFi 1010 with temperature sensors
- Infrastructure: Docker containers with Traefik load balancer

1.4 Documentation Structure

- Getting Started Setup guides and quick start
- Development Guidelines, API docs, and troubleshooting
- Architecture Documentation Technical design using arc42 template

Ready to contribute? Start with our contributing guide or jump into the build setup.

September 2, 2025

♣ DianaTin23, deadmade, deadmade

2. Getting Started

2.1 Contribute & Build

2.1.1 How do I contribute as a developer?

READ THIS GUIDE BEFORE CONTRIBUTING

Since our project is secured by two pre-commit hocks, it is important to set up the project correctly before contributing.

This is done as followed:

Clone the project

```
git clone https://github.com/deadmade/IsoPruefi.git
```

Make sure you have installed the following packages globally.

- Python: Needed for MkDocs
- Node Package Manager: Used to install needed dependencies for pre-commit hooks
- . NET 9.0 SDK: Used for our Rest-API
- Docker

After you've cloned the repo make sure to install all needed packages for the hooks via:

npm i

and run:

npm run init

Now it should be configured *

To get the development environment up and running, follow these steps:

1. Open a terminal, navigate to the IsoPrüfi directory, and run:

```
docker compose up
```

1. Once the containers are running, create an admin token for InfluxDB:

```
docker exec -it influxdb influxdb3 create token --admin
```

- 1. Copy the generated token string.
- 2. Create a config.json file at the following location:

```
IsoPruefi/isopruefi-docker/influx/explorer/config
```

1. Add the following content to <code>config.json,replacing "your-token-here"</code> with the copied token:

```
{
    "DEFAULT_INFLUX_SERVER": "http://host.docker.internal:8181",
    "DEFAULT_INFLUX_DATABASE": "IsoPrüfi",
    "DEFAULT_API_TOKEN": "your-token-here",
    "DEFAULT_SERVER_NAME": "IsoPrüfi"
}
```

- 1. Run dotnet user-secrets set "Influx:InfluxDBToken" "" -project
- 2. Paste InfluxDBToken in secrets.env
- 3. Restart the Containers

2.1.2 Arduino Set Up

Hardware

- MKR WiFi 1010
- Analog Devices ADT7410 Breakout
- DS3231 RTC
- SD Card Module

Software

⚠ Important: Always open the Arduino firmware folder (e.g., code/arduino/) as a PlatformIO Project (via Open Project or Pick a folder in the PlatformIO sidebar). Otherwise, dependencies from platformio.ini might not be detected and you may see false errors in the editor.

To work on the Arduino/PlatformIO part of the project:

- 1. Install the PlatformIO Extension in Visual Studio Code
- 2. Open the folder IsoPruefi/isopruefi-arduino
- 3. Build and upload the firmware using the PlatformIO toolbar or PlatformIO terminal
- 4. Make sure your board is connected and properly selected in platformio.ini

```
[env:mkrwifi1010]
platform = atmelsam
board = mkrwifi1010
framework = arduino
lib_deps =
    arduino-libraries/WiFiNINA
    adafruit/Adafruit ADT7410 Library
    adafruit/RTClib
    arduino-libraries/ArduinoMqttClient
    greiman/SdFat
    gyverlibs/UnixTime
    bblanchon/ArduinoJson@^7.4.2
```

Tips:

- PlatformIO installs the required libraries automatically on first build
- To run the programm run pio run -e mkrwifi1010 in the PlatformIO terminal
- To flash the Arudion with new code run pio run -e mkrwifi1010 --target upload in the PlatformIO terminal
- The main firmware entry point is located at src/main.cpp
- Use the Serial Monitor () to debug via USB
- To run the all unit tests run pio test -e native in the PlatformIO terminal

Happy Coding 😊

September 3, 2025

DianaTin23, deadmade, deadmade

2.2 Contributing Guide

2.2.1 Getting Started

Welcome to IsoPrüfi! We appreciate your interest in contributing. Please read this guide before making your first contribution.

Prerequisites

Before contributing, ensure you have:

- Read the build setup guide
- Completed the initial project setup
- Familiarized yourself with our development guidelines

2.2.2 Development Workflow

1. Branch Creation

Create branches following our naming convention:

```
git checkout -b [type]/[description-with-hyphens]
```

Branch Types:

- feat/ New features
- fix/ Bug fixes
- docs/ Documentation updates
- refactor/ Code refactoring
- test/ Adding/updating tests
- chore/ Maintenance tasks

Examples:

```
git checkout -b feat/temperature-alerts
git checkout -b fix/mqtt-connection-timeout
git checkout -b docs/api-reference-update
```

2. Development Process

- 1. Make your changes following our coding standards
- 2. Test thoroughly run relevant tests for your changes
- 3. Update documentation if needed
- 4. Commit with conventional messages (see below)

3. Testing

Run tests before committing:

```
# Frontend tests
cd isopruefi-frontend && npm test

# Backend tests
cd isopruefi-backend && dotnet test

# Arduino tests
cd isopruefi-arduino && pio test -e native
```

4. Pre-commit Hooks

Our project uses pre-commit hooks that automatically run:

- Commit message validation (Conventional Commits)

- Code formatting checks
- Basic linting

If hooks fail, fix the issues before committing again.

2.2.3 Commit Message Standards

We follow Conventional Commits:

```
<type>: <description>
[optional body]
[optional footer]
```

Allowed Types:

- feat New features
- fix Bug fixes
- docs Documentation changes
- style Code style/formatting
- refactor Code refactoring
- test Adding/updating tests
- chore Maintenance tasks
- ci CI/CD changes
- perf Performance improvements
- revert Reverting commits

Examples:

```
feat: add temperature alert notifications

fix: resolve MQTT connection timeout issue

docs: update API reference with new endpoints

refactor: improve error handling in sensor module
```

2.2.4 Pull Request Process

1. Create Pull Request

- 1. Push your branch to GitHub
- 2. Open a Pull Request against the current sprint branch
- 3. Use a descriptive title following commit conventions
- 4. Fill out the PR template completely

2. PR Requirements

Your PR must:

- Pass all automated checks (CI/CD)
- Have clear, descriptive commit messages
- Include tests for new functionality
- Update relevant documentation
- Be reviewed by at least one maintainer

3. Review Process

- · Address all reviewer feedback
- Keep PR scope focused and manageable
- Squash commits if requested

• Ensure CI passes before merge

2.2.5 Documentation

When to Update Documentation

- Adding new features or APIs
- Changing existing behavior
- Fixing bugs that affect user experience
- · Adding configuration options

Documentation Types

• API changes: Update api-reference.md

• Setup changes: Update build.md or docker-dev.md

· Architecture changes: Update relevant arc42 sections

· New features: Update main documentation

2.2.6 Issue Reporting

Bug Reports

Include:

- Clear description of the problem
- Steps to reproduce
- Expected vs actual behavior
- Environment details (OS, Docker version, etc.)
- Relevant log output

Feature Requests

Include:

- Clear description of the proposed feature
- Use case/problem it solves
- Suggested implementation approach
- Consider alternatives

2.2.7 Getting Help

• Technical questions: Open a GitHub issue

• Setup problems: Check troubleshooting guide

• Architecture questions: Reference arc42 documentation

2.2.8 Code of Conduct

- Be respectful and constructive
- Focus on the code, not the person
- Help others learn and improve
- Keep discussions technical and relevant

Thank you for contributing to IsoPrüfi! Your efforts help make the project better for everyone.

September 2, 2025

deadmade

3. Development

3.1 Guidelines / Conventions

3.1.1 General Formatting Guidelines

- Preserve Settings: Follow the existing project formatting rules.
- · Automatic Formatting: Regularly use Rider's automatic formatting function (Ctrl+Alt+L).

3.1.2 Code Layout

- Line Length: Maximum of 120 characters per line.
- · Indented Blocks: Use tabs or 4 spaces for indentation (depending on project settings).
- Blank Lines: Use blank lines to separate logical code blocks.

3.1.3 Indentation and Spacing

- · Indent Blocks: Always use 4 spaces per indentation level.
- Braces: Opening braces on the same line as the statement, closing braces on a new line.
- Operator Spacing: Add spaces around operators like +, -, *, /, =, ==, etc.
- Commas and Semicolons: Add a space after commas and semicolons, not before.

3.1.4 Naming Conventions

- Classes and Methods: Use PascalCase.
- · Variables and Fields: Use camelCase.
- Constants: Use SCREAMING_SNAKE_CASE.

3.1.5 Documentation Comments

If not obvious, or the method is more than 5 lines, it should be commented.

- Single-line Comments: Use // for single-line comments
- Multi-line Comments: Use /* ... */ for multi-line comments.
- Documentation Comments: Use /// for documentation comments.

Documentation Comments (C#)

We use the xml documentation convention by Microsoft

In short: You can use the following tags structures in your documentation comment to specify properties of the following code:

- <summary>Your code summary</summary>
- <param name="str">Describe parameter.</param>: Usage may also be nested within summary
- <code>Use a codeblock within</code>
- <example>Put a example here</example>

There are plenty more tags. You can even reference other doc segments.

If you need other tags, take a look here

Doxygen (C/C++)

For Arduino and C++ code, we use Doxygen style comments:

- /// Brief description
- /** Detailed description */
- @param name Description of parameter
- @return Description of return value
- @code ... @endcode for code blocks
- @example ... for examples

More tags: Doxygen documentation

TypeDoc (TypeScript)

For TypeScript, we use TypeDoc style comments:

- /** Summary of the function or class */
- @param name Description of parameter
- @returns Description of return value
- @example Example usage
- @see Reference to related code or docs

More tags: TypeDoc tags

3.1.6 Usings and Namespaces

- Sorting: Sort usings alphabetically and group by system namespace.
- Removing: Remove unused usings.
- Namespace: A file should contain a single namespace.

3.1.7 Error Handling

• Exceptions: Always catch specific exceptions when possible.

3.1.8 Unit Tests

Every method should be unit testable and have a unit test for it.

3.1.9 Commit Messages

How should my commit messages look like?

Our repo follows the Conventional Commits guidelines.

Allowed commit types are specified as following:

- feat -> Introduces a new features
- fix -> Fixes a bug
- docs -> Updates on the docs
- chore -> Updates a grunt task; no-production code change
- style -> Formatting code style (missing semicolon, prettier execution, etc)
- refactor -> Refactoring existing code e.g. renaming a variable, reworking a function
- ci -> Cl Tasks e.g. adding a hook
- test -> Adding new tests, refactoring tests, deleting old tests
- revert -> Revert old commits
- perf -> Performance related refactoring, without functional changes

3.1.10 Branch Naming

Your branche names should follow this style:

[commit-type]/[topic-of-branch-seperated-by-hyphen]

F.e. if you want to introduce a new cool type of button your branch should have the name:

feat/cool-new-button

August 31, 2025

♣ DianaTin23, deadmade, deadmade

3.2 Testing Strategy

3.2.1 Overview

IsoPrüfi uses a multi-layered testing approach across all components to ensure reliability and quality.

3.2.2 Test Types

Unit Tests

- Backend: NUnit framework with FluentAssertions and Moq for mocking
- $\bullet \ \textbf{Arduino} : \textbf{Unity framework for embedded testing on native platform}$
- Frontend: Not currently implemented

Integration Tests

- · Backend: Full API testing with real database connections
- · Load Tests: Performance testing with realistic data loads

3.2.3 Running Tests

Backend Tests

```
cd isopruefi-backend
dotnet test UnitTests/UnitTests.csproj
dotnet test IntegrationTests/IntegrationTests.csproj
dotnet test LoadTests/LoadTests.csproj
```

Arduino Tests

```
cd isopruefi-arduino
pio test -e native
```

3.2.4 CI/CD Testing

All tests run automatically on GitHub Actions:

- Unit tests on every push
- Integration tests on pull requests
- Docker build tests for deployment verification

September 3, 2025

deadmade

3.3 API Reference

3.3.1 Interactive API Documentation

The IsoPrüfi REST API provides endpoints for managing temperature data, sensors, and system health monitoring. Explore the interactive API documentation below to test endpoints directly.

3.3.2 Overview

Base URL:

- Development: https://backend.localhost

- Production: https://aicon.dhbw-heidenheim.de:5001/backend

API Version: v1

3.3.3 Authentication

The API uses JWT Bearer token authentication for protected endpoints.

Getting Started

- 1. Login using /v1/Authentication/Login to get your access token
- 2. Add token to requests using the Authorization: Bearer <token> header
- 3. Refresh tokens using /v1/Authentication/Refresh when they expire

3.3.4 Key Features

Temperature Data Management

- Real-time data retrieval from multiple sensor sources
- Time-range filtering with ISO 8601 timestamps
- Unit conversion between Celsius and Fahrenheit
- · Data quality validation with plausibility checks

Sensor Configuration

- MQTT topic management for sensor integration
- · Location-based sensor mapping (North, South, East, West)
- · Dynamic sensor registration and configuration

User Management

- · Role-based access control (Admin/User roles)
- · Password management and user registration
- · JWT token refresh for seamless authentication

Location Services

- · Postal code management for weather data integration
- · Geographic coordinate mapping
- · External weather service integration

3.3.5 Error Handling

The API uses standard HTTP status codes and returns detailed error information:

```
{
  "type": "https://tools.ietf.org/html/rfc7231#section-6.5.1",
  "title": "Bad Request",
  "status": 400,
  "detail": "Start time must be before end time",
  "instance": "/api/v1/TemperatureData/GetTemperature"
}
```

September 2, 2025

deadmade

3.4 API V1

September 2, 2025

deadmade

3.5 Troubleshooting Guide

3.5.1 Common Setup Issues

Docker Container Problems

CONTAINERS WON'T START

Problem: docker compose up fails or containers exit immediately

Solutions:

- Check Docker daemon is running: docker info
- Clear Docker cache: docker system prune -a
- Check logs: docker compose logs [service-name]
- Ensure sufficient resources: Check Docker Desktop settings (minimum 4GB RAM recommended)

INFLUXDB CONNECTION ERRORS

Problem: Cannot connect to InfluxDB or token issues

Solutions:

- Verify InfluxDB is running: docker ps | grep influxdb
- 2. Check InfluxDB logs: docker compose logs influxdb3
- 3. Recreate admin token:

docker exec -it influxdb influxdb3 create token --admin

- 4. Update config.json with new token in isopruefi-docker/influx/explorer/config/
- 5. Restart affected services: docker compose restart

TRAEFIK ROUTING ISSUES

Problem: *.localhost domains not accessible

Solutions:

- Verify Traefik dashboard: https://traefik.localhost:8432
- Check container labels in docker-compose.yml
- Clear browser cache/cookies for localhost domains
- Try different browser or incognito mode
- On Windows: Add entries to C:\Windows\System32\drivers\etc\hosts:

127.0.0.1 traefik.localhost 127.0.0.1 frontend.localhost 127.0.0.1 backend.localhost 127.0.0.1 grafana.localhost 127.0.0.1 explorer.localhost

- Check Traefik logs: docker compose logs traefik

DATABASE CONNECTION ISSUES

Problem: Services can't connect to PostgreSQL or InfluxDB

Solutions:

- Verify database containers are healthy: docker compose ps
- Check connection strings in environment files
- Test PostgreSQL connection:

docker exec -it postgres psql -U Isopruefi -d Isopruefi

- Test InfluxDB connection:

```
curl -H "Authorization: Token <paste-token-here>" http://localhost:8181/health
```

- Check network connectivity: docker network 1s and docker network inspect [network-name]

Arduino/PlatformIO Issues

PLATFORMIO BUILD FAILURES

Problem: Library dependencies not found or compilation errors

Solutions:

1. Clean and rebuild:

```
pio run -e mkrwifi1010 -t clean
pio lib update
pio run -e mkrwifi1010
```

- 2. Update PlatformIO: pio upgrade
- 3. Update platform/libraries: pio pkg update
- 4. Verify platformio.ini configuration
- 5. Check VS Code opened correct folder (isopruefi-arduino/)
- 6. Clear PlatformIO cache: pio system prune

SERIAL CONNECTION PROBLEMS

Problem: Cannot upload code or monitor serial output

Solutions:

- Check USB cable and connection (try different cable/port)
- Verify correct board selected in PlatformIO
- Close other applications using the serial port (Arduino IDE, other serial monitors)
- Reset board manually and try upload again
- Check device permissions (Linux/macOS): sudo usermod -a -G dialout \$USER (logout/login required)
- Windows: Update/reinstall board drivers
- Try different baud rate: pio device monitor --baud 115200

MQTT CONNECTION FAILURES

Problem: Arduino can't connect to MQTT broker

Solutions:

- Verify WiFi credentials in secrets.h
- Check MQTT broker is accessible:

```
# Test MQTT broker connection
mosquitto_pub -h your-mqtt-broker -p 1883 -t test/topic -m "test message"
```

- Verify network connectivity from Arduino location
- Check MQTT topic/payload format matches backend expectations
- Enable serial debugging in Arduino code to see connection attempts
- Try different MQTT broker (for testing): mqtt://test.mosquitto.org

Development Environment

NPM/NODE.JS ISSUES

Problem: npm install fails or pre-commit hooks don't work

Solutions:

- Update Node.js to latest LTS version (18.x or 20.x recommended)
- Clear npm cache: npm cache clean --force
- Delete node_modules/ and reinstall: rm -rf node_modules package-lock.json && npm install
- Check node/npm versions: node --version && npm --version
- Try yarn instead of npm: yarn install
- On Windows: Run as Administrator if permission issues persist
- Check proxy settings: npm config get proxy and npm config get https-proxy

.NET BUILD FAILURES

Problem: Backend won't compile or test failures

Solutions

- 1. Verify .NET 9.0 SDK: dotnet --version
- 2. Restore packages: dotnet restore
- 3. Clear build cache: dotnet clean && dotnet build
- ${\bf 4.} \ {\bf Check\ database\ connection\ strings\ in\ appsettings.json}$
- 5. Verify user secrets are set:

dotnet user-secrets list --project isopruefi-backend/MQTT-Receiver-Worker

6. Update NuGet packages: dotnet add package Microsoft.AspNetCore.App

FRONTEND BUILD ISSUES

Problem: React/TypeScript compilation errors

Solutions:

- Clear Vite cache: rm -rf node_modules/.vite
- Update dependencies: npm update
- Check TypeScript configuration: Verify tsconfig.json settings
- Restart Vite dev server: npm run dev
- Check for TypeScript errors: npm run type-check

3.5.2 Performance Issues

High resource usage

Problem: Docker containers consuming too much CPU/RAM

Solutions:

- Monitor usage: docker stats
- Increase Docker resource limits in Docker Desktop settings
- Check for memory leaks in application logs
- Optimize database queries and indexing
- Consider limiting container resources:

```
deploy:
resources:
limits:
memory: 512M
cpus: '0.5'
```

Slow web interface

Problem: Frontend loading slowly or timeouts

Solutions:

- Check browser developer tools (Network tab for slow requests)
- Verify backend API response times: docker compose logs isopruefi-backend-api-1

- Clear browser cache and data for localhost domains
- Check for JavaScript errors in browser console
- Verify network connectivity between frontend and backend containers
- Test API endpoints directly: curl -k https://backend.localhost/api/health

Database performance issues

Problem: Slow queries or connection timeouts

Solutions:

- Monitor database logs: docker compose logs postgres or docker compose logs influxdb3
- Check database indexes and query optimization
- Verify sufficient database resources
- Consider connection pooling configuration
- Monitor disk space: docker system df

3.5.3 Data Issues

Missing sensor data

Problem: Temperature readings not appearing in the system

Solutions

1. Check MQTT message flow:

```
# Check MQTT receiver logs
docker compose logs isopruefi-mqtt-receiver-1
# Test MQTT connection
mosquitto_sub -h your-mqtt-broker -t "isopruefi/+/temperature"
```

- 2. Verify InfluxDB data:
- Access InfluxDB Explorer: https://explorer.localhost
- Check database and bucket configuration
- Verify data retention policies
- 3. Check Arduino serial monitor for sensor readings and MQTT publish attempts
- 4. Verify SD card fallback is working (if offline)
- 5. Check sensor wiring and power supply
- 6. Validate MQTT message format matches backend expectations

Incorrect timestamps

Problem: Data shows wrong time or timezone issues

Solutions:

- Verify Arduino RTC module battery level
- Check system timezone settings on host machine
- Synchronize time (Linux/macOS): sudo ntpdate -s time.nist.gov
- Verify UTC handling in backend code
- Check Docker container timezone: docker exec -it container_name date
- Configure timezone in containers if needed:

```
environment:
- TZ=Europe/Berlin
```

Data duplication

Problem: Duplicate sensor readings in database

Solutions:

- Check MQTT QoS settings and message deduplication
- Verify unique constraints in database schema
- Review SD card fallback and sync logic
- Check for network reconnection issues causing retransmission
- Implement idempotent writes in backend services

3.5.4 Production Environment Issues

Health check failures

Problem: Services failing health checks in production

Solutions:

- Check service logs: docker compose logs [service-name]
- Verify health check endpoints are accessible:

```
docker exec -it container_name curl localhost:8080/health
```

- Adjust health check intervals and timeouts if needed
- Check service dependencies are ready before health checks start
- Verify network connectivity between containers

SSL/TLS certificate issues

Problem: HTTPS certificates not working in production

Solutions:

- Check Traefik certificate configuration
- Verify domain name resolution
- Check certificate expiry: openssl s_client -connect your-domain:443 -showcerts
- Review Traefik logs for certificate acquisition errors
- Ensure proper DNS configuration for domain validation

Load balancer issues

Problem: Requests not properly distributed across service instances

Solutions:

- Verify Traefik configuration and service labels
- Check service discovery and registration
- Monitor load balancing metrics in Grafana
- Test individual service instances directly
- Review sticky session configuration if needed

3.5.5 Getting Help

Log Analysis Priority

Always check logs in this order:

```
# 1. Check overall container status
docker compose ps

# 2. Check specific service logs
docker compose logs [service-name] --tail=100 -f

# 3. Check application logs inside containers
docker exec -it [container-name] tail -f /var/log/app.log
```

```
# 4. Check system logs
journalctl -u docker (Linux)
```

Debug Information Checklist

When reporting issues, include:

- Error messages and complete stack traces
- Environment info: OS, Docker version, available resources
- Docker version: docker --version && docker compose --version
- Container status: docker compose ps
- System resources: docker stats output
- Recent changes to configuration files
- Steps to reproduce the problem consistently

Essential Debug Commands

```
# Container inspection
docker inspect [container-name]
docker compose config # Validate compose file

# Network debugging
docker network ls
docker network inspect [network-name]

# Volume inspection
docker volume ls
docker volume inspect [volume-name]

# Resource monitoring
docker stats --no-stream
df -h # Check disk space

# Service-specific debugging
curl -k https://service.localhost/health
netstat -tulpn | grep :PORT
```

Common Log Locations

- Backend API: docker compose logs isopruefi-backend-api-1
- Frontend: Browser developer console + docker compose logs isopruefi-frontend-1
- Arduino: PlatformIO serial monitor (pio device monitor)
- MQTT: docker compose logs isopruefi-mqtt-receiver-1
- Databases: docker compose logs postgres or docker compose logs influxdb3
- · System logs:
- Linux: journalctl -u docker
- macOS: Console.app or log show --predicate 'process == "Docker"'
- Windows: Event Viewer (Application logs)

For additional help beyond this guide:

- Check the contributing guide for development workflow issues
- Review API documentation for backend integration problems
- Create an issue in the GitHub repository with debug information

September 2, 2025

≜ deadmade

4. Code Documentation

4.1 Assembly Rest-API

4.1.1 Namespace Rest_API

• Program

4.1.2 Namespace Rest_API.Controllers

- AuthenticationController
- LocationController
- TemperatureDataController
- TopicController
- UserInfoController

4.1.3 Namespace Rest_API.Helper

- HealthCheck
- StringTools

4.1.4 Namespace Rest_API.Models

- ChangePassword
- JwtToken
- Login
- Register
- Roles
- SensorData
- TemperatureData
- TemperatureDataOverview

4.1.5 Namespace Rest_API.Seeder

• SeedUser

4.1.6 Namespace Rest_API.Services.Auth

- AuthenticationService
- IAuthenticationService

4.1.7 Namespace Rest_API.Services.Temp

- ITempService
- TempService

4.1.8 Namespace Rest_API.Services.Token

- ITokenService
- TokenService

4.1.9 Namespace Rest_API.Services.User

- IUserService
- UserService

September 3, 2025

*

4.2 Assembly MQTT-Receiver-Worker

4.2.1 Namespace MQTT_Receiver_Worker

- HealthCheck
- NullMetaListConverter
- Program
- Worker

4.2.2 Namespace MQTT_Receiver_Worker.MQTT

- Connection
- MqttHealthCheck
- Receiver

4.2.3 Namespace MQTT_Receiver_Worker.MQTT.Interfaces

- IConnection
- IReceiver

4.2.4 Namespace MQTT_Receiver_Worker.MQTT.Models

- TempSensorMeta
- TempSensorReading

September 3, 2025

*

4.3 Assembly MQTT-Sender

4.3.1 Namespace MQTT_Sender

• Connection

September 3, 2025

**

4.4 Assembly Database

4.4.1 Namespace Database. Entity Framework

- ApplicationDbContext
- 4.4.2 Namespace Database.EntityFramework.Enums
 - SensorType
- 4.4.3 Namespace Database.EntityFramework.Models
 - ApiUser
 - CoordinateMapping
 - TokenInfo
 - TopicSetting

4.4.4 Namespace Database. Migrations

RefactorMigrations

4.4.5 Namespace Database.Repository.CoordinateRepo

- CoordinateRepo
- ICoordinateRepo

4.4.6 Namespace Database.Repository.InfluxRepo

- IInfluxRepo
- InfluxRetryService

4.4.7 Namespace Database.Repository.InfluxRepo.Influx

- InfluxHealthCheck
- InfluxRepo

4.4.8 Namespace Database.Repository.InfluxRepo.InfluxCache

- CachedInfluxHealthCheck
- CachedInfluxRepo

4.4.9 Namespace Database.Repository.SettingsRepo

- ISettingsRepo
- SettingsRepo

4.4.10 Namespace Database.Repository.TokenRepo

- ITokenRepo
- TokenRepo

September 3, 2025

**

4.5 Assembly UnitTests

4.5.1 Namespace UnitTests.Controllers

- AuthenticationControllerTests
- LocationControllerTests
- $\bullet \, \mathsf{TemperatureDataControllerTests}$
- TopicControllerTests
- UserInfoControllerTests

4.5.2 Namespace UnitTests.MqttReceiver

- ConnectionTests
- NullMetaListConverterTests
- ReceiverTests
- $\bullet \ Temp Sensor Reading Tests$
- WorkerTests

4.5.3 Namespace UnitTests.Repositories

- InfluxRepoTests
- SettingsRepoTests

4.5.4 Namespace UnitTests.Services

- AuthenticationServiceTests
- TempServiceTests
- TokenServiceTests
- UserServiceTests

September 3, 2025

**

4.6 Assembly IntegrationTests

4.6.1 Namespace IntegrationTests

- HealthCheckIntegrationTests
- SimpleAuthTest
- StartupIntegrationTests

4.6.2 Namespace IntegrationTests.ApiClient

- ApiException
- · ApiException<TResult>
- ApiUser
- AuthenticationClient
- ChangePassword
- CoordinateMapping
- FileResponse
- IdentityUser
- IdentityUserOfString
- JwtToken
- LocationClient
- Login
- ProblemDetails
- Register
- SensorData
- SensorType
- TemperatureData
- TemperatureDataClient
- TemperatureDataOverview
- TopicClient
- TopicSetting
- UserInfoClient

4.6.3 Namespace IntegrationTests.Controllers

- $\bullet \, Authentication Controller Integration Tests$
- ${\bf \cdot} Location Controller Integration Tests$
- $\bullet \ Temperature Data Controller Integration Tests$
- $\bullet \ Topic Controller Integration Tests\\$
- $\bullet \ UserInfoControllerIntegrationTests$

4.6.4 Namespace IntegrationTests.Infrastructure

ApiClientTestBase

IntegrationTestBase

4.6.5 Namespace IntegrationTests.WeatherWorker

• WeatherWorkerIntegrationTests

September 3, 2025

<u>::</u>:

4.7 Assembly LoadTests

4.7.1 Namespace IntegrationTests.ApiClient

- ApiException
- ApiException<TResult>
- ApiUser
- AuthenticationClient
- ChangePassword
- CoordinateMapping
- FileResponse
- IdentityUser
- IdentityUserOfString
- JwtToken
- LocationClient
- Login
- ProblemDetails
- Register
- SensorData
- SensorType
- TemperatureData
- TemperatureDataClient
- TemperatureDataOverview
- TopicClient
- TopicSetting
- UserInfoClient

4.7.2 Namespace LoadTests.Infrastructure

- AuthHelper
- LoadTestBase
- TokenResponse

4.7.3 Namespace LoadTests.Seeder

- InfluxSeeder
- SensorSeeder

4.7.4 Namespace LoadTests.Tests

- BasicInfrastructureTest
- MqttSensorLoadTest
- RestApiLoadTest

September 3, 2025

4.8 Contents pages

- Global contents
- Structures
- Files
- Modules
- Directories

4.9 Index pages

- Global index
- Structures
- Files
- Modules
- Directories

September 3, 2025

**

4.10 Frontend

isopruefi-frontend v1.0.0

4.10.1 isopruefi-frontend v1.0.0

Modules

- api/api-client
- api/clients
- App
- auth/AuthForm
- auth/SignIn
- auth/SignUp
- components/Navbar
- components/ProtectedRoute
- components/Weather
- main
- pages/AdminPage
- pages/UserPage
- pages/Welcome
- utils/authApi
- utils/config
- utils/tokenHelpers

September 3, 2025

4.10.2 Api

api-client

isopruefi-frontend v1.0.0

isopruefi-frontend / api/api-client

API/API-CLIENT

Enumerations

SensorType

Classes

- ApiException
- ApiUser
- AuthenticationClient
- ChangePassword
- CoordinateMapping
- IdentityUser
- IdentityUserOfString
- JwtToken
- LocationClient
- Login
- ProblemDetails
- Register
- SensorData
- TemperatureData
- TemperatureDataClient
- TemperatureDataOverview
- TopicClient
- TopicSetting
- UserInfoClient

Interfaces

- FileResponse
- IApiUser
- IChangePassword
- ICoordinateMapping
- IIdentityUser
- IIdentityUserOfString
- IJwtToken
- ILogin
- IProblemDetails
- IRegister
- ISensorData

- ITemperatureData
- ITemperatureDataOverview
- ITopicSetting

September 3, 2025

*

CLASSES

message

message: string

isopruefi-frontend v1.0.0

```
isopruefi-frontend / api/api-client / ApiException
 Class: ApiException
  Defined in: api/api-client.ts:1846
Extends
    • Error
Constructors Constructor
     \textbf{new ApiException} (\, \texttt{message} \, , \, \texttt{status} \, , \, \texttt{response} \, , \, \texttt{headers} \, , \, \texttt{result} \, ) \\ \vdots \, \, \texttt{ApiException}
  Defined in: api/api-client.ts:1853
Parameters message
   string
status
   number
response
   string
headers result
   any
Returns
   ApiException
Overrides
   Error.constructor
Properties headers
     headers: object
  Defined in: api/api-client.ts:1850
Index Signature
  [key: string]: any
isApiException
     protected isApiException: boolean = true
  Defined in: api/api-client.ts:1863
```

Defined in: api/api-client.ts:1847 Overrides Error.message response response: string Defined in: api/api-client.ts:1849 result result: any Defined in: api/api-client.ts:1851 status status: number Defined in: api/api-client.ts:1848 Methods isApiException() static isApiException(obj): obj is ApiException Defined in: api/api-client.ts:1865 Parameters obj any Returns obj is ApiException September 3, 2025

:

Inherited from

```
isopruefi-frontend / api/api-client / ApiUser
Class: ApiUser
  Defined in: api/api-client.ts:1811
  Represents an application user in the system
Extends
   • IdentityUser
Implements
   • IApiUser
Constructors Constructor
    new ApiUser( data? ): ApiUser
  Defined in: api/api-client.ts:1813
Parameters data?
  IApiUser
Returns
  ApiUser
Overrides
  IdentityUser.constructor
Properties accessFailedCount?
    optional accessFailedCount: number
  Defined in: api/api-client.ts:1688
  Gets or sets the number of failed login attempts for the current user.
Implementation of
  IApiUser.accessFailedCount
Inherited from
  IdentityUser.accessFailedCount
concurrencyStamp?
    optional concurrencyStamp: string
  Defined in: api/api-client.ts:1676
  A random value that must change whenever a user is persisted to the store
Implementation of
  IApiUser.concurrencyStamp
```

${\tt IdentityUser.concurrencyStamp}$

IApiUser.lockoutEnabled

```
email?
  optional email: string
  Defined in: api/api-client.ts:1666
  Gets or sets the email address for this user.
Implementation of
  IApiUser.email
Inherited from
  IdentityUser.email
emailConfirmed?
    optional emailConfirmed: boolean
  Defined in: api/api-client.ts:1670
  Gets or sets a flag indicating if a user has confirmed their email address.
Implementation of
  IApiUser . emailConfirmed
Inherited from
  IdentityUser.emailConfirmed
id?
    optional id: string
  Defined in: api/api-client.ts:1660
  Gets or sets the primary key for this user.
Implementation of
  IApiUser.id
Inherited from
  IdentityUser.id
lockoutEnabled?
    optional lockoutEnabled: boolean
  Defined in: api/api-client.ts:1686
  Gets or sets a flag indicating if the user could be locked out.
Implementation of
```

Inherited from

 ${\tt IdentityUser.lockoutEnabled}$

lockoutEnd?

optional lockoutEnd: Date

Defined in: api/api-client.ts:1684

Gets or sets the date and time, in UTC, when any user lockout ends.

Implementation of

IApiUser.lockoutEnd

Inherited from

IdentityUser.lockoutEnd

normalizedEmail?

optional normalizedEmail: string

Defined in: api/api-client.ts:1668

Gets or sets the normalized email address for this user.

Implementation of

IApiUser.normalizedEmail

Inherited from

 ${\tt IdentityUser.normalizedEmail}$

normalizedUserName?

optional normalizedUserName: string

Defined in: api/api-client.ts:1664

Gets or sets the normalized user name for this user.

Implementation of

 ${\tt IApiUser.normalizedUserName}$

Inherited from

IdentityUser.normalizedUserName

passwordHash?

optional passwordHash: string

Defined in: api/api-client.ts:1672

Gets or sets a salted and hashed representation of the password for this user.

Implementation of

IApiUser.passwordHash

Inherited from

IdentityUser.passwordHash

phoneNumber?

optional phoneNumber: string

Defined in: api/api-client.ts:1678

Gets or sets a telephone number for the user.

Implementation of

IApiUser.phoneNumber

Inherited from

IdentityUser.phoneNumber

phoneNumberConfirmed?

optional phoneNumberConfirmed: boolean

Defined in: api/api-client.ts:1680

Gets or sets a flag indicating if a user has confirmed their telephone address.

Implementation of

 ${\tt IApiUser.phoneNumberConfirmed}$

Inherited from

 ${\tt IdentityUser.phoneNumberConfirmed}$

securityStamp?

optional securityStamp: string

Defined in: api/api-client.ts:1674

A random value that must change whenever a users credentials change (password changed, login removed)

Implementation of

IApiUser.securityStamp

Inherited from

IdentityUser.securityStamp

twoFactorEnabled?

optional twoFactorEnabled: boolean

Defined in: api/api-client.ts:1682

Gets or sets a flag indicating if two factor authentication is enabled for this user.

```
Implementation of
  IApiUser.twoFactorEnabled
Inherited from
  IdentityUser .twoFactorEnabled
userName?
  optional userName: string
  Defined in: api/api-client.ts:1662
  Gets or sets the user name for this user.
Implementation of
  IApiUser.userName
Inherited from
  IdentityUser.userName
Methods init()
  init( _data? ): void
  Defined in: api/api-client.ts:1817
Parameters _data?
  any
Returns
  void
Overrides
  IdentityUser.init
toJSON()
  toJSON( data? ): any
  Defined in: api/api-client.ts:1828
Parameters data?
  any
Returns
  any
Overrides
  IdentityUser.toJSON
fromJS()
```

static **fromJS**(data): ApiUser

Defined in: api/api-client.ts:1821

Parameters data

any

Returns

ApiUser

Overrides

IdentityUser.fromJS

September 3, 2025

*

```
isopruefi-frontend / api/api-client / AuthenticationClient
 Class: AuthenticationClient
  Defined in: api/api-client.ts:10
Constructors Constructor
  new AuthenticationClient( baseUrl?, http?): AuthenticationClient
  Defined in: api/api-client.ts:15
Parameters baseUrl?
   string
http? fetch Returns
  AuthenticationClient
Properties jsonParseReviver
    protected jsonParseReviver: undefined | (key, value) => any = undefined
  Defined in: api/api-client.ts:13
Methods login()
  login(input): Promise \< FileResponse >
  Defined in: api/api-client.ts:25
  Authenticates a user and returns a JWT token for API access.
Parameters input
  Login
  The login credentials containing username and password.
Returns
  Promise \< FileResponse >
  Authentication successful. Returns JWT access token and refresh token.
processLogin()
    protected processLogin( response ): Promise \< FileResponse >
  Defined in: api/api-client.ts:45
Parameters response
  Response
Returns
  Promise \< FileResponse >
```

processRefresh() protected processRefresh(response): Promise \< void > Defined in: api/api-client.ts:155 Parameters response Response Returns Promise \< void > processRegister() protected processRegister(response): Promise \< void > Defined in: api/api-client.ts:91 Parameters response Response Returns Promise \< void > refresh() refresh(token): Promise \< void > Defined in: api/api-client.ts:136 Refreshes an expired JWT access token using a valid refresh token. Parameters token JwtToken The JWT token object containing both the expired access token and valid refresh token. Returns Promise \< void > Token refresh successful. Returns new access and refresh tokens. register() register(input): Promise \< void > Defined in: api/api-client.ts:72 Registers a new user in the system. Admin access required. Parameters input Register The registration data containing username and password for the new user.

Returns

Promise \< void >

User registered successfully.

September 3, 2025

**

IChangePassword.userId

```
isopruefi-frontend / api/api-client / ChangePassword
 Class: ChangePassword
  Defined in: api/api-client.ts:1600
  Represents a request to change a user's password.
Implements
   • IChangePassword
Constructors Constructor
    new ChangePassword( data? ): ChangePassword
  Defined in: api/api-client.ts:1611
Parameters data?
  IChangePassword
Returns
  ChangePassword
Properties currentPassword?
    optional currentPassword: string
  Defined in: api/api-client.ts:1606
  Gets or sets the current password of the user.
Implementation of
  IChangePassword.currentPassword
newPassword?
     optional newPassword: string
  Defined in: api/api-client.ts:1609
  Gets or sets the new password to be set for the user.
Implementation of
  IChangePassword . newPassword
userId?
     optional userld: string
  Defined in: api/api-client.ts:1603
  Gets or sets the unique identifier of the user whose password is to be changed.
Implementation of
```

Methods init() init(_data?): void Defined in: api/api-client.ts:1620 Parameters _data? any Returns void toJSON() toJSON(data?): any Defined in: api/api-client.ts:1635 Parameters data? any Returns any fromJS() static **fromJS**(data): ChangePassword Defined in: api/api-client.ts:1628 Parameters data any Returns ChangePassword September 3, 2025 :

```
isopruefi-frontend / api/api-client / CoordinateMapping
```

Class: CoordinateMapping

Defined in: api/api-client.ts:1508

Stores geographic coordinates associated with postalcodes, including the time the mapping was used.

Implements

• ICoordinateMapping

Constructors Constructor

new CoordinateMapping(data?): CoordinateMapping

Defined in: api/api-client.ts:1528

Parameters data?

ICoordinateMapping

Returns

CoordinateMapping

Properties lastUsed?

optional lastUsed: Date

Defined in: api/api-client.ts:1523

Gets or sets the time the postalcode was last entered by the user.

Implementation of

ICoordinateMapping.lastUsed

latitude?

optional latitude: number

Defined in: api/api-client.ts:1517

Gets or sets the latitude for the location.

Implementation of

ICoordinateMapping.latitude

location?

optional location: string

Defined in: api/api-client.ts:1514

Gets or sets the name of the location.

Implementation of

ICoordinateMapping.location

```
lockedUntil?
  optional lockedUntil: Date
  Defined in: api/api-client.ts:1526
  Gets or sets the time until which the entry is locked.
Implementation of
  ICoordinateMapping.lockedUntil
longitude?
    optional longitude: number
  Defined in: api/api-client.ts:1520
  Gets or sets the longitude of the location.
Implementation of
  {\tt ICoordinateMapping.longitude}
postalCode?
    optional postalCode: number
  Defined in: api/api-client.ts:1511
  Gets or sets the postalcode which is also the uniqe identifier.
Implementation of
  ICoordinateMapping.postalCode
Methods init()
  init( _data? ): void
  Defined in: api/api-client.ts:1537
Parameters _data?
  any
Returns
  void
toJSON()
  toJSON( data? ): any
  Defined in: api/api-client.ts:1555
Parameters data?
  any
Returns
```

any

fromJS()

static fromJS(data): CoordinateMapping

Defined in: api/api-client.ts:1548

Parameters data

any

Returns

CoordinateMapping

September 3, 2025

:

```
isopruefi-frontend / api/api-client / IdentityUser
 Class: IdentityUser
  Defined in: api/api-client.ts:1782
  The default implementation of IdentityUser`1 which uses a string as a primary key.
Extends
   • IdentityUserOfString
Extended by

    ApiUser

Implements
   • IIdentityUser
Constructors Constructor
    new IdentityUser( data? ): IdentityUser
  Defined in: api/api-client.ts:1784
Parameters data?
  IIdentityUser
Returns
  IdentityUser
Overrides
  IdentityUserOfString.constructor
Properties accessFailedCount?
    optional accessFailedCount: number
  Defined in: api/api-client.ts:1688
  Gets or sets the number of failed login attempts for the current user.
Implementation of
  IIdentityUser .accessFailedCount
Inherited from
  IdentityUserOfString.accessFailedCount
concurrencyStamp?
```

optional concurrencyStamp: string

Defined in: api/api-client.ts:1676

A random value that must change whenever a user is persisted to the store

Implementation of

```
{\bf IIdentity User}. {\bf concurrency Stamp}
Inherited from
  IdentityUserOfString.concurrencyStamp
email?
  optional email: string
  Defined in: api/api-client.ts:1666
  Gets or sets the email address for this user.
Implementation of
  IIdentityUser.email
Inherited from
  IdentityUserOfString.email
emailConfirmed?
  optional emailConfirmed: boolean
  Defined in: api/api-client.ts:1670
  Gets or sets a flag indicating if a user has confirmed their email address.
Implementation of
  IIdentityUser . emailConfirmed
Inherited from
  {\tt IdentityUserOfString}\ .\ {\tt emailConfirmed}
id?
    optional id: string
  Defined in: api/api-client.ts:1660
  Gets or sets the primary key for this user.
Implementation of
  IIdentityUser.id
Inherited from
  IdentityUserOfString.id
lockoutEnabled?
```

optional lockoutEnabled: boolean

Defined in: api/api-client.ts:1686

Gets or sets a flag indicating if the user could be locked out.

IIdentityUser . lockoutEnabled

Inherited from

IdentityUserOfString.lockoutEnabled

lockoutEnd?

optional lockoutEnd: Date

Defined in: api/api-client.ts:1684

Gets or sets the date and time, in UTC, when any user lockout ends.

Implementation of

IIdentityUser.lockoutEnd

Inherited from

 ${\tt IdentityUserOfString.lockoutEnd}$

normalizedEmail?

optional normalizedEmail: string

Defined in: api/api-client.ts:1668

Gets or sets the normalized email address for this user.

Implementation of

IIdentityUser . normalizedEmail

Inherited from

IdentityUserOfString.normalizedEmail

normalizedUserName?

optional normalizedUserName: string

Defined in: api/api-client.ts:1664

Gets or sets the normalized user name for this user.

Implementation of

IIdentityUser.normalizedUserName

Inherited from

 ${\tt IdentityUserOfString.normalizedUserName}$

passwordHash?

optional passwordHash: string

Defined in: api/api-client.ts:1672

Gets or sets a salted and hashed representation of the password for this user.

Implementation of

IIdentityUser.passwordHash

Inherited from

IdentityUserOfString.passwordHash

phoneNumber?

optional phoneNumber: string

Defined in: api/api-client.ts:1678

Gets or sets a telephone number for the user.

Implementation of

IIdentityUser.phoneNumber

Inherited from

IdentityUserOfString.phoneNumber

phoneNumberConfirmed?

optional phoneNumberConfirmed: boolean

Defined in: api/api-client.ts:1680

Gets or sets a flag indicating if a user has confirmed their telephone address.

Implementation of

 ${\bf IIdentity User}\ .\ phone {\bf Number Confirmed}$

Inherited from

IdentityUserOfString.phoneNumberConfirmed

securityStamp?

optional securityStamp: string

Defined in: api/api-client.ts:1674

A random value that must change whenever a users credentials change (password changed, login removed)

Implementation of

IIdentityUser.securityStamp

Inherited from

 ${\tt IdentityUserOfString.securityStamp}$

twoFactorEnabled?

optional twoFactorEnabled: boolean

```
Defined in: api/api-client.ts:1682
  Gets or sets a flag indicating if two factor authentication is enabled for this user.
Implementation of
  IIdentityUser.twoFactorEnabled
Inherited from
  {\tt IdentityUserOfString.twoFactorEnabled}
userName?
     optional userName: string
  Defined in: api/api-client.ts:1662
  Gets or sets the user name for this user.
Implementation of
  IIdentityUser.userName
Inherited from
  IdentityUserOfString.userName
Methods init()
  init( _data? ): void
  Defined in: api/api-client.ts:1788
Parameters _data?
   any
Returns
  void
Overrides
  IdentityUserOfString.init
toJSON()
  toJSON( data? ): any
  Defined in: api/api-client.ts:1799
Parameters data?
  any
Returns
   any
Overrides
  IdentityUserOfString.toJSON
```

fromJS()

static **fromJS**(data): IdentityUser

Defined in: api/api-client.ts:1792

Parameters data

any

Returns

IdentityUser

Overrides

IdentityUserOfString.fromJS

September 3, 2025

*

isopruefi-frontend / api/api-client / IdentityUserOfString

Class: IdentityUserOfString

Defined in: api/api-client.ts:1658

Represents a user in the identity system

Extended by

• IdentityUser

Implements

• IIdentityUserOfString

Constructors Constructor

new IdentityUserOfString(data?): IdentityUserOfString

Defined in: api/api-client.ts:1690

Parameters data?

IIdentityUserOfString

Returns

IdentityUserOfString

Properties accessFailedCount?

optional accessFailedCount: number

Defined in: api/api-client.ts:1688

Gets or sets the number of failed login attempts for the current user.

Implementation of

 ${\bf IIdentity User Of String}\ .\ access Failed Count$

concurrencyStamp?

optional concurrencyStamp: string

Defined in: api/api-client.ts:1676

A random value that must change whenever a user is persisted to the store

Implementation of

 ${\bf IIdentity User Of String}\ .\ concurrency Stamp$

email?

optional email: string

Defined in: api/api-client.ts:1666

Gets or sets the email address for this user.

 ${\bf IIdentity User Of String}\ .\ {\bf email}$

emailConfirmed?

optional emailConfirmed: boolean

Defined in: api/api-client.ts:1670

Gets or sets a flag indicating if a user has confirmed their email address.

Implementation of

 ${\bf IIdentity User Of String}\ .\ {\bf email Confirmed}$

id?

optional id: string

Defined in: api/api-client.ts:1660

Gets or sets the primary key for this user.

Implementation of

IIdentityUserOfString.id

lockoutEnabled?

optional lockoutEnabled: boolean

Defined in: api/api-client.ts:1686

Gets or sets a flag indicating if the user could be locked out.

Implementation of

IIdentityUserOfString.lockoutEnabled

lockoutEnd?

optional lockoutEnd: Date

Defined in: api/api-client.ts:1684

Gets or sets the date and time, in UTC, when any user lockout ends.

Implementation of

IIdentityUserOfString.lockoutEnd

normalizedEmail?

optional normalizedEmail: string

Defined in: api/api-client.ts:1668

Gets or sets the normalized email address for this user.

IIdentityUserOfString.normalizedEmail

normalizedUserName?

optional normalizedUserName: string

Defined in: api/api-client.ts:1664

Gets or sets the normalized user name for this user.

Implementation of

 ${\bf IIdentity User Of String} \ . \ normalized {\bf User Name}$

passwordHash?

optional passwordHash: string

Defined in: api/api-client.ts:1672

Gets or sets a salted and hashed representation of the password for this user.

Implementation of

 ${\bf IIdentity User Of String.password Hash}\\$

phoneNumber?

optional phoneNumber: string

Defined in: api/api-client.ts:1678

Gets or sets a telephone number for the user.

Implementation of

IIdentityUserOfString.phoneNumber

phoneNumberConfirmed?

optional phoneNumberConfirmed: boolean

Defined in: api/api-client.ts:1680

Gets or sets a flag indicating if a user has confirmed their telephone address.

Implementation of

 ${\bf IIdentity User Of String\:.\: phone Number Confirmed\:}$

securityStamp?

optional securityStamp: string

Defined in: api/api-client.ts:1674

A random value that must change whenever a users credentials change (password changed, login removed)

```
{\tt IIdentityUser0fString.securityStamp}
```

```
twoFactorEnabled?
```

optional twoFactorEnabled: boolean

Defined in: api/api-client.ts:1682

Gets or sets a flag indicating if two factor authentication is enabled for this user.

Implementation of

 ${\bf IIdentity User Of String\ .\ two Factor Enabled}$

userName?

optional userName: string

Defined in: api/api-client.ts:1662

Gets or sets the user name for this user.

Implementation of

IIdentityUserOfString.userName

Methods init()

init(_data?): void

Defined in: api/api-client.ts:1699

Parameters _data?

any

Returns

void

toJSON()

toJSON(data?): any

Defined in: api/api-client.ts:1726

Parameters data?

any

Returns

any

fromJS()

static fromJS(data): IdentityUserOfString

Defined in: api/api-client.ts:1719

Parameters data

any

Returns

IdentityUserOfString

September 3, 2025

:

```
isopruefi-frontend / api/api-client / JwtToken
 Class: JwtToken
  Defined in: api/api-client.ts:1130
  Represents a JWT token and its associated refresh token and metadata.
Implements
   • IJwtToken
Constructors Constructor
    new JwtToken( data? ): JwtToken
  Defined in: api/api-client.ts:1147
Parameters data?
  IJwtToken
Returns
  JwtToken
Properties createdDate?
    optional createdDate: Date
  Defined in: api/api-client.ts:1142
  Gets or sets the creation date and time of the JWT token.
Implementation of
  IJwtToken.createdDate
expiryDate?
    optional expiryDate: Date
  Defined in: api/api-client.ts:1139
  Gets or sets the expiry date and time of the JWT token.
Implementation of
  IJwtToken.expiryDate
refreshToken?
     optional refreshToken: string
  Defined in: api/api-client.ts:1136
  Gets or sets the refresh token string.
Implementation of
  IJwtToken.refreshToken
```

```
roles?
  optional roles: string[
  Defined in: api/api-client.ts:1145
  Gets or sets the user roles associated with the JWT token.
Implementation of
  IJwtToken.roles
token?
    optional token: string
  Defined in: api/api-client.ts:1133
  Gets or sets the JWT access token string.
Implementation of
  IJwtToken . token
Methods init()
  init( _data? ): void
  Defined in: api/api-client.ts:1156
Parameters _data?
  any
Returns
  void
toJSON()
  toJSON( data? ): any
  Defined in: api/api-client.ts:1177
Parameters data?
  any
Returns
  any
fromJS()
  static fromJS(data): JwtToken
  Defined in: api/api-client.ts:1170
Parameters data
  any
```

Returns

JwtToken

September 3, 2025

**

```
isopruefi-frontend / api/api-client / LocationClient
 Class: LocationClient
  Defined in: api/api-client.ts:189
Constructors Constructor
  new LocationClient( baseUrl? , http?): LocationClient
  Defined in: api/api-client.ts:194
Parameters baseUrl?
   string
http? fetch Returns
   LocationClient
Properties jsonParseReviver
    protected jsonParseReviver: undefined | (key, value) => any = undefined
  Defined in: api/api-client.ts:192
Methods getAllPostalcodes()
  getAllPostalcodes(): Promise \< FileResponse >
  Defined in: api/api-client.ts:203
  Retrieves all saved locations.
Returns
   Promise \< FileResponse >
  A list of all postalcodes; otherwise, NotFound.
insertLocation()
    insertLocation( postalcode? ): Promise \< FileResponse >
  Defined in: api/api-client.ts:246
  Checks for existence of location and if necessary inserts new location.
Parameters postalcode?
   number
  (optional) Defines the location.
Returns
   Promise \< FileResponse >
  Ok if successful; otherwise, an error response.
```

```
processGetAllPostalcodes()
     protected processGetAllPostalcodes( response ): Promise \< FileResponse >
  Defined in: api/api-client.ts:219
Parameters response
  Response
Returns
  Promise \< FileResponse >
processInsertLocation()
     protected processInsertLocation( response ): Promise \< FileResponse >
  Defined in: api/api-client.ts:266
Parameters response
  Response
Returns
  Promise \< FileResponse >
processRemovePostalcode()
     protected processRemovePostalcode( response ): Promise \< void >
  Defined in: api/api-client.ts:312
Parameters response
  Response
Returns
  Promise \< void >
removePostalcode()
    removePostalcode( postalCode? ): Promise \< void >
  Defined in: api/api-client.ts:293
  Deletes location from the database.
Parameters postalCode?
  number
  (optional) Postalcode
Returns
  Promise \< void >
  Ok if successful; otherwise, an error response.
```

September 3, 2025

:

```
isopruefi-frontend / api/api-client / Login
Class: Login
  Defined in: api/api-client.ts:966
  Represents the login credentials for a user.
Implements
   • ILogin
Constructors Constructor
  new Login(data?): Login
  Defined in: api/api-client.ts:974
Parameters data?
  ILogin
Returns
  Login
Properties password
  password: string
  Defined in: api/api-client.ts:972
  Gets or sets the password of the user.
Implementation of
  ILogin.password
userName
    userName: string
  Defined in: api/api-client.ts:969
  Gets or sets the username of the user.
Implementation of
  ILogin.userName
Methods init()
  init( _data? ): void
  Defined in: api/api-client.ts:983
Parameters _data?
  any
Returns
  void
```

toJSON()

toJSON(data?): any

Defined in: api/api-client.ts:997

Parameters data?

any

Returns

any

fromJS()

static **fromJS**(data): Login

Defined in: api/api-client.ts:990

Parameters data

any

Returns

Login

September 3, 2025

**

```
isopruefi-frontend / api/api-client / ProblemDetails
 Class: ProblemDetails
  Defined in: api/api-client.ts:1015
Implements

    IProblemDetails

Indexable
  [key:string]:any
Constructors Constructor
  new ProblemDetails( data? ): ProblemDetails
  Defined in: api/api-client.ts:1024
Parameters data?
  IProblemDetails
Returns
  ProblemDetails
Properties detail?
  optional detail: string
  Defined in: api/api-client.ts:1019
Implementation of
  IProblemDetails.detail
instance?
     optional instance: string
  Defined in: api/api-client.ts:1020
Implementation of
  IProblemDetails.instance
status?
    optional status: number
  Defined in: api/api-client.ts:1018
Implementation of
  IProblemDetails.status
```

```
title?
  optional title: string
  Defined in: api/api-client.ts:1017
Implementation of
  IProblemDetails.title
type?
    optional type: string
  Defined in: api/api-client.ts:1016
Implementation of
  IProblemDetails.type
Methods init()
    init( _data? ): void
  Defined in: api/api-client.ts:1033
Parameters _data?
  any
Returns
  void
toJSON()
  toJSON( data? ): any
  Defined in: api/api-client.ts:1054
Parameters data?
  any
Returns
  any
fromJS()
    static fromJS( data ): ProblemDetails
  Defined in: api/api-client.ts:1047
Parameters data
  any
Returns
  ProblemDetails
```

September 3, 2025

:

```
isopruefi-frontend / api/api-client / Register
Class: Register
  Defined in: api/api-client.ts:1080
  Represents the registration credentials for a new user.
Implements
   • IRegister
Constructors Constructor
    new Register (data?): Register
  Defined in: api/api-client.ts:1088
Parameters data?
  IRegister
Returns
  Register
Properties password
  password: string
  Defined in: api/api-client.ts:1086
  Gets or sets the password for the new user.
Implementation of
  IRegister . password
userName
    userName: string
  Defined in: api/api-client.ts:1083
  Gets or sets the username for the new user.
Implementation of
  IRegister.userName
Methods init()
  init( _data? ): void
  Defined in: api/api-client.ts:1097
Parameters _data?
  any
Returns
  void
```

toJSON()

toJSON(data?): any

Defined in: api/api-client.ts:1111

Parameters data?

any

Returns

any

fromJS()

static **fromJS**(data): Register

Defined in: api/api-client.ts:1104

Parameters data

any

Returns

Register

September 3, 2025

```
isopruefi-frontend / api/api-client / SensorData
 Class: SensorData
  Defined in: api/api-client.ts:1278
  Represents an overview of sensor data.
Implements
   • ISensorData
Constructors Constructor
    new SensorData( data? ): SensorData
  Defined in: api/api-client.ts:1289
Parameters data?
  ISensorData
Returns
  SensorData
Properties location?
    optional location: string
  Defined in: api/api-client.ts:1284
  Gets or sets the location of the sensor.
Implementation of
  ISensorData.location
sensorName?
     optional sensorName: string
  Defined in: api/api-client.ts:1281
  Gets or sets the name of the sensor.
Implementation of
  ISensorData.sensorName
temperatureDatas?
     optional temperatureDatas: TemperatureData []
  Defined in: api/api-client.ts:1287
  Gets or sets the temperature data of the sensor.
Implementation of
  ISensorData.temperatureDatas
```

```
Methods init()
  init( _data? ): void
  Defined in: api/api-client.ts:1298
Parameters _data?
  any
Returns
  void
toJSON()
  toJSON( data? ): any
  Defined in: api/api-client.ts:1317
Parameters data?
  any
Returns
  any
fromJS()
  static fromJS(data): SensorData
  Defined in: api/api-client.ts:1310
Parameters data
  any
Returns
  SensorData
September 3, 2025
:
```

ITemperatureData.timestamp

```
isopruefi-frontend / api/api-client / TemperatureData
 Class: TemperatureData
  Defined in: api/api-client.ts:1344
  Represents a single temperature data point with timestamp and value.
Implements
   • ITemperatureData
Constructors Constructor
    new TemperatureData( data? ): TemperatureData
  Defined in: api/api-client.ts:1355
Parameters data?
  ITemperatureData
Returns
  TemperatureData
Properties plausibility?
    optional plausibility: string
  Defined in: api/api-client.ts:1353
  Gets or sets the plausibility of the temperature data.
Implementation of
  ITemperatureData.plausibility
temperature?
     optional temperature: number
  Defined in: api/api-client.ts:1350
  Gets or sets the temperature value.
Implementation of
  ITemperatureData.temperature
timestamp?
     optional timestamp: Date
  Defined in: api/api-client.ts:1347
  Gets or sets the timestamp of the temperature measurement.
Implementation of
```

Methods init() init(_data?): void Defined in: api/api-client.ts:1364 Parameters _data? any Returns void toJSON() toJSON(data?): any Defined in: api/api-client.ts:1379 Parameters data? any Returns any fromJS() static **fromJS**(data): TemperatureData Defined in: api/api-client.ts:1372 Parameters data any Returns TemperatureData September 3, 2025 :

```
isopruefi-frontend / api/api-client / TemperatureDataClient
 Class: TemperatureDataClient
  Defined in: api/api-client.ts:328
Constructors Constructor
    new TemperatureDataClient( baseUrl?, http?): TemperatureDataClient
  Defined in: api/api-client.ts:333
Parameters baseUrl?
   string
http? fetch Returns
  TemperatureDataClient
Properties jsonParseReviver
    protected jsonParseReviver: undefined | ( key , value ) => any = undefined
  Defined in: api/api-client.ts:331
Methods getTemperature()
  getTemperature( start?, end?, place?, isFahrenheit?): Promise \< TemperatureDataOverview >
  Defined in: api/api-client.ts:346
  Retrieves comprehensive temperature data for a specified time range and location.
Parameters start?
  Date
  (optional) Start date and time for the data range (ISO 8601 format).
end?
  Date
  (optional) End date and time for the data range (ISO 8601 format).
place?
  (optional) Location name for external weather data (e.g., "Berlin", "Munich").
isFahrenheit?
  boolean
  (optional) Optional. If true, converts all temperatures to Fahrenheit. Default is false (Celsius).
Returns
  Promise \< TemperatureDataOverview >
  Successfully retrieved temperature data. Returns comprehensive temperature overview.
```

processGetTemperature()

protected processGetTemperature(response): Promise \< TemperatureDataOverview >

Defined in: api/api-client.ts:378

Parameters response

Response

Returns

Promise \< TemperatureDataOverview >

September 3, 2025

**

Returns

void

```
isopruefi-frontend / api/api-client / TemperatureDataOverview
 Class: TemperatureDataOverview
  Defined in: api/api-client.ts:1212
  Represents an overview of temperature data for different locations.
Implements
   • ITemperatureDataOverview
Constructors Constructor
    new TemperatureDataOverview( data? ): TemperatureDataOverview
  Defined in: api/api-client.ts:1220
Parameters data?
  ITemperatureDataOverview
Returns
  TemperatureDataOverview
Properties sensorData?
    optional sensorData: SensorData [
  Defined in: api/api-client.ts:1215
  Gets or sets the list of Sensor data for the inside temperature.
Implementation of
  ITemperature Data Overview.sensor Data\\
temperatureOutside?
     optional temperatureOutside: TemperatureData [
  Defined in: api/api-client.ts:1218
  Gets or sets the list of temperature data for the outside location.
Implementation of
  ITemperatureDataOverview.temperatureOutside
Methods init()
  init( _data? ): void
  Defined in: api/api-client.ts:1229
Parameters _data?
  any
```

toJSON()

toJSON(data?): any

Defined in: api/api-client.ts:1251

Parameters data?

any

Returns

any

fromJS()

static **fromJS**(data): TemperatureDataOverview

Defined in: api/api-client.ts:1244

Parameters data

any

Returns

TemperatureDataOverview

September 3, 2025

Promise \< any >

```
isopruefi-frontend / api/api-client / TopicClient
 Class: TopicClient
  Defined in: api/api-client.ts:422
Constructors Constructor
  new TopicClient( baseUrl?, http?): TopicClient
  Defined in: api/api-client.ts:427
Parameters baseUrl?
   string
http? fetch Returns
   TopicClient
Properties jsonParseReviver
    protected jsonParseReviver: undefined | (key, value) => any = undefined
  Defined in: api/api-client.ts:425
Methods createTopic()
  createTopic( topicSetting ): Promise \< any >
  Defined in: api/api-client.ts:563
  Creates a new MQTT topic configuration for sensor monitoring.
Parameters topicSetting
   TopicSetting
  The complete topic setting configuration to create.
Returns
   Promise \< any >
  Topic setting created successfully. Returns the new topic ID.
deleteTopic()
    deleteTopic( topicSetting ): Promise \< any >
  Defined in: api/api-client.ts:701
  Deletes an existing MQTT topic setting from the system.
Parameters topicSetting
   TopicSetting
  The topic setting object identifying the configuration to delete.
Returns
```

Successfully deleted the topic setting.

```
getAllSensorTypes()
  getAllSensorTypes(): Promise \< string []>
  Defined in: api/api-client.ts:499
  Retrieves all available sensor types from the system.
Returns
  Promise \< string []>
  Successfully retrieved the list of sensor types.
getAllTopics()
  getAllTopics(): Promise \< TopicSetting []>
  Defined in: api/api-client.ts:436
  Retrieves all configured MQTT topic settings from the system.
Returns
  Promise \< TopicSetting []>
  Successfully retrieved all topic settings.
processCreateTopic()
     protected processCreateTopic( response ): Promise \< any >
  Defined in: api/api-client.ts:583
Parameters response
  Response
Returns
  Promise \< any >
processDeleteTopic()
     protected processDeleteTopic( response ): Promise \< any >
  Defined in: api/api-client.ts:721
Parameters response
  Response
Returns
  Promise \< any >
```

```
processGetAllSensorTypes()
     protected processGetAllSensorTypes( response ): Promise \< string []>
  Defined in: api/api-client.ts:515
Parameters response
  Response
Returns
  Promise \< string []>
processGetAllTopics()
     protected processGetAllTopics( response ): Promise \< TopicSetting []>
  Defined in: api/api-client.ts:452
Parameters response
  Response
Returns
  Promise \< TopicSetting []>
processUpdateTopic()
    protected processUpdateTopic( response ): Promise \< any >
  Defined in: api/api-client.ts:652
Parameters response
  Response
Returns
  Promise \< any >
updateTopic()
    updateTopic( topicSetting ): Promise \< any >
  Defined in: api/api-client.ts:632
  Updates an existing MQTT topic setting in the system.
Parameters topicSetting
  TopicSetting
  The topic setting object containing updated configuration.
Returns
  Promise \< any >
  Successfully updated the topic setting.
```

September 3, 2025

:

Implementation of

ITopicSetting.defaultTopicPath

```
isopruefi-frontend / api/api-client / TopicSetting
 Class: TopicSetting
  Defined in: api/api-client.ts:1402
  Represents the settings for a specific MQTT topic, including default path, group, and sensor information.
Implements

    ITopicSetting

Constructors Constructor
    new TopicSetting( data? ): TopicSetting
  Defined in: api/api-client.ts:1431
Parameters data?
  ITopicSetting
Returns
  TopicSetting
Properties coordinateMapping?
    optional coordinateMapping: CoordinateMapping
  Defined in: api/api-client.ts:1411
  Navigation property for the related CoordinateMapping entity.
Implementation of
  ITopicSetting.coordinateMapping
coordinateMappingId?
     optional coordinateMappingId: number
  Defined in: api/api-client.ts:1408
  Gets or sets the foreign key for the related CoordinateMapping entity.
Implementation of
  ITopicSetting.coordinateMappingId
defaultTopicPath?
     optional defaultTopicPath: string
  Defined in: api/api-client.ts:1414
  Gets or sets the default MQTT topic path for this setting.
```

```
groupId?
  optional groupId: number
  Defined in: api/api-client.ts:1417
  Gets or sets the group identifier associated with this topic setting.
Implementation of
  ITopicSetting.groupId
hasRecovery?
     optional hasRecovery: boolean
  Defined in: api/api-client.ts:1429
  Gets or sets a value indicating whether this topic setting has recovery enabled.
Implementation of
  ITopicSetting.hasRecovery
sensorLocation?
     optional sensorLocation: string
  Defined in: api/api-client.ts:1426
  Gets or sets the location of the sensor.
Implementation of
  ITopicSetting.sensorLocation
sensorName?
     optional sensorName: string
  Defined in: api/api-client.ts:1423
  Gets or sets the name of the sensor.
Implementation of
  ITopicSetting.sensorName
sensorTypeEnum?
    optional sensorTypeEnum: SensorType
  Defined in: api/api-client.ts:1420
  Gets or sets the type of sensor (e.g., temperature, humidity).
Implementation of
  ITopicSetting.sensorTypeEnum
```

```
topicSettingId?
    optional topicSettingId: number
  Defined in: api/api-client.ts:1405
  Gets or sets the unique identifier for the TopicSetting entity.
Implementation of
  ITopicSetting.topicSettingId
Methods init()
  init( _data? ): void
  Defined in: api/api-client.ts:1440
Parameters _data?
  any
Returns
  void
toJSON()
  toJSON( data? ): any
  Defined in: api/api-client.ts:1461
Parameters data?
  any
Returns
  any
fromJS()
  static fromJS(data): TopicSetting
  Defined in: api/api-client.ts:1454
Parameters data
  any
Returns
  TopicSetting
September 3, 2025
```

*

Promise \< FileResponse >

```
isopruefi-frontend / api/api-client / UserInfoClient
 Class: UserInfoClient
  Defined in: api/api-client.ts:766
Constructors Constructor
  new UserInfoClient( baseUrl?, http?): UserInfoClient
  Defined in: api/api-client.ts:771
Parameters baseUrl?
   string
http? fetch Returns
  UserInfoClient
Properties jsonParseReviver
    protected jsonParseReviver: undefined | (key, value) => any = undefined
  Defined in: api/api-client.ts:769
Methods changePassword()
    changePassword( input ): Promise \< FileResponse >
  Defined in: api/api-client.ts:828
  Changes the password for a user.
Parameters input
  ChangePassword
  The change password request containing user ID, current password, and new password.
Returns
  Promise \< FileResponse >
  Ok if successful; otherwise, an error response.
changeUser()
  changeUser( user ): Promise \< FileResponse >
  Defined in: api/api-client.ts:875
  Updates user information.
Parameters user
  ApiUser
  The user object with updated information.
Returns
```

Ok if successful; otherwise, an error response.

Defined in: api/api-client.ts:895

```
deleteUser()
  deleteUser( userId? ): Promise \< FileResponse >
  Defined in: api/api-client.ts:922
  Deletes a user by their unique identifier.
Parameters userId?
  string
  (optional) The unique identifier of the user to delete.
Returns
  Promise \< FileResponse >
  Ok if successful; otherwise, an error response.
getUserById()
  getUserById( userId? ): Promise \< FileResponse >
  Defined in: api/api-client.ts:781
  Retrieves a user by their unique identifier.
Parameters userId?
  string
  (optional) The unique identifier of the user.
Returns
  Promise \< FileResponse >
  The user information if found; otherwise, NotFound.
processChangePassword()
    protected processChangePassword( response ): Promise \< FileResponse >
  Defined in: api/api-client.ts:848
Parameters response
  Response
Returns
  Promise \< FileResponse >
processChangeUser()
  protected processChangeUser( response ): Promise \< FileResponse >
```

Parameters response

Response

Returns

Promise \< FileResponse >

processDeleteUser()

protected processDeleteUser(response): Promise \< FileResponse >

Defined in: api/api-client.ts:942

Parameters response

Response

Returns

Promise \< FileResponse >

processGetUserById()

protected processGetUserById(response): Promise \< FileResponse >

Defined in: api/api-client.ts:801

Parameters response

Response

Returns

Promise \< FileResponse >

September 3, 2025

**

ENUMERATIONS

:

isopruefi-frontend v1.0.0

```
isopruefi-frontend / api/api-client / SensorType
 Enumeration: SensorType
  Defined in: api/api-client.ts:1590
  Represents the different types of sensors available in the system.
Enumeration Members Co2
    Co2: 4
  Defined in: api/api-client.ts:1595
Hum
    Hum: 2
  Defined in: api/api-client.ts:1593
Ikea
  Ikea: 3
  Defined in: api/api-client.ts:1594
Mic
  Mic: 5
  Defined in: api/api-client.ts:1596
Spl
    Spl: 1
  Defined in: api/api-client.ts:1592
Temp
  Temp: 0
  Defined in: api/api-client.ts:1591
September 3, 2025
```

INTERFACES

isopruefi-frontend v1.0.0

isopruefi-frontend / api/api-client / FileResponse

Interface: FileResponse

Defined in: api/api-client.ts:1839

Properties data

data: Blob

Defined in: api/api-client.ts:1840

fileName?

optional fileName: string

Defined in: api/api-client.ts:1842

headers?

optional **headers**: object

Defined in: api/api-client.ts:1843

Index Signature

[name: string]: any

status

status: number

Defined in: api/api-client.ts:1841

September 3, 2025

<u>::</u>:

isopruefi-frontend / api/api-client / IApiUser

Interface: IApiUser

Defined in: api/api-client.ts:1836

Represents an application user in the system

Extends

• IIdentityUser

Properties accessFailedCount?

optional accessFailedCount: number

Defined in: api/api-client.ts:1778

Gets or sets the number of failed login attempts for the current user.

Inherited from

IIdentityUser .accessFailedCount

concurrencyStamp?

optional concurrencyStamp: string

Defined in: api/api-client.ts:1766

A random value that must change whenever a user is persisted to the store

Inherited from

 ${\bf IIdentity User}\ .\ {\bf concurrency Stamp}$

email?

optional **email**: string

Defined in: api/api-client.ts:1756

Gets or sets the email address for this user.

Inherited from

IIdentityUser.email

emailConfirmed?

optional emailConfirmed: boolean

Defined in: api/api-client.ts:1760

Gets or sets a flag indicating if a user has confirmed their email address.

Inherited from

IIdentityUser.emailConfirmed

```
id?
```

optional id: string

Defined in: api/api-client.ts:1750

Gets or sets the primary key for this user.

Inherited from

IIdentityUser.id

lockoutEnabled?

optional lockoutEnabled: boolean

Defined in: api/api-client.ts:1776

Gets or sets a flag indicating if the user could be locked out.

Inherited from

IIdentityUser . lockoutEnabled

lockoutEnd?

optional lockoutEnd: Date

Defined in: api/api-client.ts:1774

Gets or sets the date and time, in UTC, when any user lockout ends.

Inherited from

IIdentityUser.lockoutEnd

normalizedEmail?

optional normalizedEmail: string

Defined in: api/api-client.ts:1758

Gets or sets the normalized email address for this user.

Inherited from

 ${\bf IIdentity User.normalized Email}\\$

normalizedUserName?

optional normalizedUserName: string

Defined in: api/api-client.ts:1754

Gets or sets the normalized user name for this user.

Inherited from

IIdentityUser.normalizedUserName

passwordHash?

optional passwordHash: string

Defined in: api/api-client.ts:1762

Gets or sets a salted and hashed representation of the password for this user.

Inherited from

 ${\bf IIdentity User\,.\,password Hash}$

phoneNumber?

optional phoneNumber: string

Defined in: api/api-client.ts:1768

Gets or sets a telephone number for the user.

Inherited from

IIdentityUser.phoneNumber

phoneNumberConfirmed?

optional phoneNumberConfirmed: boolean

Defined in: api/api-client.ts:1770

Gets or sets a flag indicating if a user has confirmed their telephone address.

Inherited from

 ${\bf IIdentity User}\ .\ phone {\bf Number Confirmed}$

securityStamp?

optional securityStamp: string

Defined in: api/api-client.ts:1764

A random value that must change whenever a users credentials change (password changed, login removed)

Inherited from

 ${\bf IIdentity User}\ .\ {\bf security Stamp}$

twoFactorEnabled?

optional twoFactorEnabled: boolean

Defined in: api/api-client.ts:1772

Gets or sets a flag indicating if two factor authentication is enabled for this user.

Inherited from

 ${\bf IIdentity User}\ .\ {\bf two Factor Enabled}$

userName?

optional userName: string

Defined in: api/api-client.ts:1752

Gets or sets the user name for this user.

Inherited from

IIdentityUser.userName

September 3, 2025

**

isopruefi-frontend / api/api-client / IChangePassword

Interface: IChangePassword

Defined in: api/api-client.ts:1645

Represents a request to change a user's password.

Properties currentPassword?

optional currentPassword: string

Defined in: api/api-client.ts:1651

Gets or sets the current password of the user.

newPassword?

optional newPassword: string

Defined in: api/api-client.ts:1654

Gets or sets the new password to be set for the user.

userId?

optional userld: string

Defined in: api/api-client.ts:1648

Gets or sets the unique identifier of the user whose password is to be changed.

September 3, 2025

*

isopruefi-frontend / api/api-client / ICoordinateMapping

Interface: ICoordinateMapping

Defined in: api/api-client.ts:1568

Stores geographic coordinates associated with postalcodes, including the time the mapping was used.

Properties lastUsed?

optional lastUsed: Date

Defined in: api/api-client.ts:1583

Gets or sets the time the postalcode was last entered by the user.

latitude?

optional latitude: number

Defined in: api/api-client.ts:1577

Gets or sets the latitude for the location.

location?

optional location: string

Defined in: api/api-client.ts:1574

Gets or sets the name of the location.

lockedUntil?

optional lockedUntil: Date

Defined in: api/api-client.ts:1586

Gets or sets the time until which the entry is locked.

longitude?

optional longitude: number

Defined in: api/api-client.ts:1580

Gets or sets the longitude of the location.

postalCode?

optional postalCode: number

Defined in: api/api-client.ts:1571

Gets or sets the postalcode which is also the uniqe identifier.

September 3, 2025

:

isopruefi-frontend / api/api-client / IldentityUser

Interface: IldentityUser

Defined in: api/api-client.ts:1807

The default implementation of IdentityUser`1 which uses a string as a primary key.

Extends

• IIdentityUserOfString

Extended by

• IApiUser

Properties accessFailedCount?

optional accessFailedCount: number

Defined in: api/api-client.ts:1778

Gets or sets the number of failed login attempts for the current user.

Inherited from

 ${\bf IIdentity User Of String.\,access Failed Count}$

concurrencyStamp?

optional concurrencyStamp: string

Defined in: api/api-client.ts:1766

A random value that must change whenever a user is persisted to the store

Inherited from

 ${\bf IIdentity User Of String}\ .\ concurrency Stamp$

email?

optional email: string

Defined in: api/api-client.ts:1756

Gets or sets the email address for this user.

Inherited from

 ${\bf IIdentity User Of String}\ .\ {\bf email}$

emailConfirmed?

optional emailConfirmed: boolean

Defined in: api/api-client.ts:1760

Gets or sets a flag indicating if a user has confirmed their email address.

Inherited from

 ${\bf IIdentity User Of String}\:.\:email Confirmed$

id?

optional id: string

Defined in: api/api-client.ts:1750

Gets or sets the primary key for this user.

Inherited from

 ${\bf IIdentity User Of String.id}\\$

lockoutEnabled?

optional lockoutEnabled: boolean

Defined in: api/api-client.ts:1776

Gets or sets a flag indicating if the user could be locked out.

Inherited from

 ${\bf IIdentity User Of String.lockout Enabled}$

lockoutEnd?

optional lockoutEnd: Date

Defined in: api/api-client.ts:1774

Gets or sets the date and time, in UTC, when any user lockout ends.

Inherited from

IIdentityUserOfString.lockoutEnd

normalizedEmail?

optional normalizedEmail: string

Defined in: api/api-client.ts:1758

Gets or sets the normalized email address for this user.

Inherited from

IIdentityUserOfString.normalizedEmail

normalizedUserName?

optional normalizedUserName: string

Defined in: api/api-client.ts:1754

Gets or sets the normalized user name for this user.

Inherited from

IIdentityUserOfString.normalizedUserName

passwordHash?

optional passwordHash: string

Defined in: api/api-client.ts:1762

Gets or sets a salted and hashed representation of the password for this user.

Inherited from

 ${\bf IIdentity User Of String.\,password Hash}$

phoneNumber?

optional phoneNumber: string

Defined in: api/api-client.ts:1768

Gets or sets a telephone number for the user.

Inherited from

 ${\bf IIdentity User Of String\:.\: phone Number\:}$

phoneNumberConfirmed?

optional phoneNumberConfirmed: boolean

Defined in: api/api-client.ts:1770

Gets or sets a flag indicating if a user has confirmed their telephone address.

Inherited from

IIdentityUserOfString.phoneNumberConfirmed

securityStamp?

optional securityStamp: string

Defined in: api/api-client.ts:1764

A random value that must change whenever a users credentials change (password changed, login removed)

Inherited from

IIdentityUserOfString.securityStamp

twoFactorEnabled?

optional twoFactorEnabled: boolean

Defined in: api/api-client.ts:1772

Gets or sets a flag indicating if two factor authentication is enabled for this user.

Inherited from

 ${\bf IIdentity User Of String.two Factor Enabled}$

userName?

optional userName: string

Defined in: api/api-client.ts:1752

Gets or sets the user name for this user.

Inherited from

IIdentityUserOfString.userName

September 3, 2025

:

isopruefi-frontend / api/api-client / IldentityUserOfString

Interface: IldentityUserOfString

Defined in: api/api-client.ts:1748

Represents a user in the identity system

Extended by

• IIdentityUser

Properties accessFailedCount?

optional accessFailedCount: number

Defined in: api/api-client.ts:1778

Gets or sets the number of failed login attempts for the current user.

concurrencyStamp?

optional concurrencyStamp: string

Defined in: api/api-client.ts:1766

A random value that must change whenever a user is persisted to the store

email?

optional email: string

Defined in: api/api-client.ts:1756

Gets or sets the email address for this user.

emailConfirmed?

optional emailConfirmed: boolean

Defined in: api/api-client.ts:1760

Gets or sets a flag indicating if a user has confirmed their email address.

id?

optional id: string

Defined in: api/api-client.ts:1750

Gets or sets the primary key for this user.

lockoutEnabled?

optional lockoutEnabled: boolean

Defined in: api/api-client.ts:1776

Gets or sets a flag indicating if the user could be locked out.

lockoutEnd?

optional lockoutEnd: Date

Defined in: api/api-client.ts:1774

Gets or sets the date and time, in UTC, when any user lockout ends.

normalizedEmail?

optional normalizedEmail: string

Defined in: api/api-client.ts:1758

Gets or sets the normalized email address for this user.

normalizedUserName?

optional normalizedUserName: string

Defined in: api/api-client.ts:1754

Gets or sets the normalized user name for this user.

passwordHash?

optional passwordHash: string

Defined in: api/api-client.ts:1762

Gets or sets a salted and hashed representation of the password for this user.

phoneNumber?

optional phoneNumber: string

Defined in: api/api-client.ts:1768

Gets or sets a telephone number for the user.

phoneNumberConfirmed?

optional phoneNumberConfirmed: boolean

Defined in: api/api-client.ts:1770

Gets or sets a flag indicating if a user has confirmed their telephone address.

securityStamp?

optional securityStamp: string

Defined in: api/api-client.ts:1764

A random value that must change whenever a users credentials change (password changed, login removed)

twoFactorEnabled?

optional twoFactorEnabled: boolean

Defined in: api/api-client.ts:1772

Gets or sets a flag indicating if two factor authentication is enabled for this user.

userName?

optional userName: string

Defined in: api/api-client.ts:1752

Gets or sets the user name for this user.

September 3, 2025

:

isopruefi-frontend / api/api-client / IJwtToken

Interface: IJwtToken

Defined in: api/api-client.ts:1193

Represents a JWT token and its associated refresh token and metadata.

Properties createdDate?

optional **createdDate**: Date

Defined in: api/api-client.ts:1205

Gets or sets the creation date and time of the JWT token.

expiryDate?

optional expiryDate: Date

Defined in: api/api-client.ts:1202

Gets or sets the expiry date and time of the JWT token.

refreshToken?

optional refreshToken: string

Defined in: api/api-client.ts:1199

Gets or sets the refresh token string.

roles?

optional roles: string[

Defined in: api/api-client.ts:1208

Gets or sets the user roles associated with the JWT token.

token?

optional token: string

Defined in: api/api-client.ts:1196

Gets or sets the JWT access token string.

September 3, 2025

isopruefi-frontend / api/api-client / ILogin

Interface: ILogin

Defined in: api/api-client.ts:1006

Represents the login credentials for a user.

Properties password

password: string

Defined in: api/api-client.ts:1012

Gets or sets the password of the user.

userName

userName: string

Defined in: api/api-client.ts:1009

Gets or sets the username of the user.

September 3, 2025

isopruefi-frontend / api/api-client / IProblemDetails

Interface: IProblemDetails

Defined in: api/api-client.ts:1069

Indexable

[key:string]:any

Properties detail?

optional **detail**: string

Defined in: api/api-client.ts:1073

instance?

optional instance: string

Defined in: api/api-client.ts:1074

status?

optional **status**: number

Defined in: api/api-client.ts:1072

title?

optional title: string

Defined in: api/api-client.ts:1071

type?

optional **type**: string

Defined in: api/api-client.ts:1070

September 3, 2025

isopruefi-frontend / api/api-client / IRegister

Interface: IRegister

Defined in: api/api-client.ts:1120

Represents the registration credentials for a new user.

Properties password

password: string

Defined in: api/api-client.ts:1126

Gets or sets the password for the new user.

userName

userName: string

Defined in: api/api-client.ts:1123

Gets or sets the username for the new user.

September 3, 2025

isopruefi-frontend / api/api-client / ISensorData

Interface: ISensorData

Defined in: api/api-client.ts:1331

Represents an overview of sensor data.

Properties location?

optional location: string

Defined in: api/api-client.ts:1337

Gets or sets the location of the sensor.

sensorName?

optional sensorName: string

Defined in: api/api-client.ts:1334

Gets or sets the name of the sensor.

temperatureDatas?

optional temperatureDatas: TemperatureData [

Defined in: api/api-client.ts:1340

Gets or sets the temperature data of the sensor.

September 3, 2025

isopruefi-frontend / api/api-client / ITemperatureData

Interface: ITemperatureData

Defined in: api/api-client.ts:1389

Represents a single temperature data point with timestamp and value.

Properties plausibility?

optional plausibility: string

Defined in: api/api-client.ts:1398

Gets or sets the plausibility of the temperature data.

temperature?

optional temperature: number

Defined in: api/api-client.ts:1395

Gets or sets the temperature value.

timestamp?

optional timestamp: Date

Defined in: api/api-client.ts:1392

Gets or sets the timestamp of the temperature measurement.

September 3, 2025

isopruefi-frontend / api/api-client / ITemperatureDataOverview

Interface: ITemperatureDataOverview

Defined in: api/api-client.ts:1268

Represents an overview of temperature data for different locations.

Properties sensorData?

optional **sensorData**: SensorData[

Defined in: api/api-client.ts:1271

Gets or sets the list of Sensor data for the inside temperature.

temperatureOutside?

optional temperatureOutside: TemperatureData [

Defined in: api/api-client.ts:1274

Gets or sets the list of temperature data for the outside location.

September 3, 2025

•••

isopruefi-frontend / api/api-client / ITopicSetting

Interface: ITopicSetting

Defined in: api/api-client.ts:1477

Represents the settings for a specific MQTT topic, including default path, group, and sensor information.

Properties coordinateMapping?

optional coordinateMapping: CoordinateMapping

Defined in: api/api-client.ts:1486

Navigation property for the related CoordinateMapping entity.

coordinateMappingId?

optional coordinateMappingId: number

Defined in: api/api-client.ts:1483

Gets or sets the foreign key for the related CoordinateMapping entity.

defaultTopicPath?

optional defaultTopicPath: string

Defined in: api/api-client.ts:1489

Gets or sets the default MQTT topic path for this setting.

groupId?

optional **groupld**: number

Defined in: api/api-client.ts:1492

Gets or sets the group identifier associated with this topic setting.

hasRecovery?

optional hasRecovery: boolean

Defined in: api/api-client.ts:1504

Gets or sets a value indicating whether this topic setting has recovery enabled.

sensorLocation?

optional sensorLocation: string

Defined in: api/api-client.ts:1501

Gets or sets the location of the sensor.

sensorName?

optional sensorName: string

Defined in: api/api-client.ts:1498

Gets or sets the name of the sensor.

sensorTypeEnum?

optional sensorTypeEnum: SensorType

Defined in: api/api-client.ts:1495

Gets or sets the type of sensor (e.g., temperature, humidity).

topicSettingId?

optional topicSettingId: number

Defined in: api/api-client.ts:1480

Gets or sets the unique identifier for the TopicSetting entity.

September 3, 2025

<u>...</u>

clients

isopruefi-frontend v1.0.0

isopruefi-frontend / api/clients

API/CLIENTS

Type Aliases

PostalLocation

Variables

- authClient
- locationClient
- tempClient
- topicClient

Functions

- addPostalLocation
- createTopic
- deleteTopic
- fetchPostalLocations
- getAllTopics
- getStoredLocationName
- ${\color{gray}\bullet} \ remove Postal Location$
- updateTopic

References

ApiException

Re-exports ApiException

TopicSetting

Re-exports TopicSetting

September 3, 2025

<u>::</u>:

FUNCTIONS

isopruefi-frontend v1.0.0

isopruefi-frontend / api/clients / addPostalLocation

Function: addPostalLocation()

addPostalLocation(postalCode): Promise \< FileResponse >

Defined in: api/clients.ts:155

Adds a new postal code location to the system.

Parameters postalCode

number

The postal code to add

Returns

Promise \< FileResponse >

Promise resolving to the API response

Throws

When the request fails or postal code already exists

September 3, 2025

:

isopruefi-frontend / api/clients / createTopic

Function: createTopic()

createTopic(topicSetting): Promise \< any >

Defined in: api/clients.ts:187

Creates a new MQTT topic configuration in the system.

Parameters topicSetting

TopicSetting

The complete topic setting configuration to create

Returns

Promise \< any >

Promise resolving to the newly created topic with assigned ID

Throws

When validation fails, topic already exists, or access is denied

September 3, 2025

isopruefi-frontend / api/clients / deleteTopic

Function: deleteTopic()

deleteTopic(topicSetting): Promise \< any >

Defined in: api/clients.ts:209

Removes an MQTT topic configuration from the system.

Parameters topicSetting

TopicSetting

The topic setting to delete (requires topicSettingId)

Returns

Promise \< any >

Promise resolving to void on successful deletion

Throws

When topic doesn't exist or access is denied

September 3, 2025

isopruefi-frontend / api/clients / fetchPostalLocations

Function: fetchPostalLocations()

fetchPostalLocations(): Promise \< object []>

Defined in: api/clients.ts:71

Fetches all postal code locations from the API and normalizes the response format.

Handles multiple possible response formats from the backend API.

Returns

Promise \< object []>

Promise resolving to an array of PostalLocation objects

Throws

When API request fails

September 3, 2025

isopruefi-frontend / api/clients / getAllTopics

Function: getAllTopics()

getAllTopics(): Promise \< TopicSetting []>

Defined in: api/clients.ts:176

Retrieves all MQTT topic settings from the system.

Returns

Promise \< TopicSetting []>

Promise resolving to an array of TopicSetting objects

Throws

When the request fails or access is denied

September 3, 2025

isopruefi-frontend / api/clients / getStoredLocationName

Function: getStoredLocationName()

getStoredLocationName(displayLocationName): string

Defined in: api/clients.ts:144

Retrieves the backend-stored location name for a display location name. Used internally to map user-friendly display names to backend identifiers.

Parameters displayLocationName

string

The display name shown to users

Returns

string

The corresponding stored location name for API calls

September 3, 2025

:

isopruefi-frontend / api/clients / removePostalLocation

Function: removePostalLocation()

removePostalLocation(postalCode): Promise \< void >

Defined in: api/clients.ts:166

Removes a postal code location from the system.

Parameters postalCode

number

The postal code to remove

Returns

Promise \< void >

Promise resolving to void on successful removal

Throws

When the request fails or postal code doesn't exist

September 3, 2025

isopruefi-frontend / api/clients / updateTopic

Function: updateTopic()

updateTopic(topicSetting): Promise \< any >

Defined in: api/clients.ts:198

 $\label{thm:local_potential} \mbox{Updates an existing MQTT topic configuration}.$

Parameters topicSetting

TopicSetting

The topic setting with updated values (must include topicSettingId)

Returns

Promise \< any >

Promise resolving to the updated topic setting

Throws

When validation fails, topic doesn't exist, or access is denied

September 3, 2025

TYPE-ALIASES

isopruefi-frontend v1.0.0

isopruefi-frontend / api/clients / PostalLocation

Type Alias: PostalLocation

PostalLocation = object

Defined in: api/clients.ts:217

Represents a postal code location with its associated name.

Used for location-based temperature data queries.

Properties locationName

locationName: string

Defined in: api/clients.ts:221

The human-readable location name

postalCode

postalCode: number

Defined in: api/clients.ts:219

The postal code number

September 3, 2025

:::

VARIABLES

isopruefi-frontend v1.0.0

isopruefi-frontend / api/clients / authClient

Variable: authClient

const authClient: AuthenticationClient

Defined in: api/clients.ts:42

Pre-configured authentication client with automatic JWT token handling.

September 3, 2025

isopruefi-frontend / api/clients / locationClient

Variable: locationClient

const locationClient: LocationClient

Defined in: api/clients.ts:52

Pre-configured location client with automatic JWT token handling.

September 3, 2025

isopruefi-frontend / api/clients / tempClient

Variable: tempClient

const tempClient: TemperatureDataClient

Defined in: api/clients.ts:47

Pre-configured temperature data client with automatic JWT token handling.

September 3, 2025

isopruefi-frontend / api/clients / topicClient

Variable: topicClient

const topicClient: TopicClient

Defined in: api/clients.ts:57

Pre-configured MQTT topic client with automatic JWT token handling.

September 3, 2025

4.10.3 Authentication

AuthForm

isopruefi-frontend v1.0.0

isopruefi-frontend / auth/AuthForm

AUTH/AUTHFORM

Functions

default

September 3, 2025

FUNCTIONS

isopruefi-frontend v1.0.0

isopruefi-frontend / auth/AuthForm / default

Function: default()

default(props): Element

Defined in: auth/AuthForm.tsx:31

Authentication form component for sign in and sign up.

Handles user input, authentication API calls, error display, and navigation.

On successful login, sets the global authentication state and navigates to the appropriate page.

On registration, shows a success message and navigates to the sign in page.

Parameters props

AuthFormProps

Component props.

Returns

Element

The rendered authentication form.

September 3, 2025

<u>::</u>:

SignIn

isopruefi-frontend v1.0.0

isopruefi-frontend / auth/SignIn

AUTH/SIGNIN

Functions

default

September 3, 2025

FUNCTIONS

isopruefi-frontend v1.0.0

isopruefi-frontend / auth/SignIn / default

Function: default()

default(): Element

Defined in: auth/SignIn.tsx:3

Returns

Element

September 3, 2025



SignUp

isopruefi-frontend v1.0.0

isopruefi-frontend / auth/SignUp

AUTH/SIGNUP

Functions

default

September 3, 2025

FUNCTIONS

isopruefi-frontend v1.0.0

isopruefi-frontend / auth/SignUp / default

Function: default()

default(): Element

Defined in: auth/SignUp.tsx:3

Returns

Element

September 3, 2025



4.10.4 Components

Navbar

isopruefi-frontend v1.0.0

isopruefi-frontend / components/Navbar

COMPONENTS/NAVBAR

September 3, 2025

ProtectedRoute

isopruefi-frontend v1.0.0

isopruefi-frontend / components/ProtectedRoute

COMPONENTS/PROTECTEDROUTE

Functions

default

September 3, 2025

isopruefi-frontend v1.0.0

isopruefi-frontend / components/ProtectedRoute / default

Function: default()

default(props): Element

Defined in: components/ProtectedRoute.tsx:15

ProtectedRoute component for role-based route protection.

Checks authentication and user role before rendering child routes.

- If authentication is not ready, shows a loading indicator.
- If user is not authenticated, redirects to the public welcome page.
- If user role is not allowed, redirects to their default page.

Parameters props

Component props.

allowed

("admin" | "user")[]

Array of allowed roles for the route.

Returns

Element

The rendered protected route or a redirect.

September 3, 2025

Weather

isopruefi-frontend v1.0.0

isopruefi-frontend / components/Weather

COMPONENTS/WEATHER

Type Aliases

WeatherEntry

Functions

TempChart

September 3, 2025

**

isopruefi-frontend v1.0.0

isopruefi-frontend / components/Weather / TempChart

Function: TempChart()

TempChart(props): Element

Defined in: components/Weather.tsx:36

Displays a temperature chart and sensor tiles for a given location.

Fetches weather and sensor data from the backend, supports filtering by time range,

and allows switching between Celsius and Fahrenheit.

Parameters props

TempChartProps = {}

Component props.

Returns

Element

The rendered temperature chart and sensor tiles.

September 3, 2025

TYPE-ALIASES

isopruefi-frontend v1.0.0

```
isopruefi-frontend / components/Weather / WeatherEntry
```

Type Alias: WeatherEntry

WeatherEntry = object

Defined in: components/Weather.tsx:13

Represents a single weather data entry for a specific timestamp.

Indexable

```
[key: string]: undefined | string | number
```

Properties t

t: number

Defined in: components/Weather.tsx:15

Epoch time in milliseconds.

tempOutside?

optional tempOutside: number

Defined in: components/Weather.tsx:17

Outside temperature value.

timestamp

timestamp: string

Defined in: components/Weather.tsx:14

ISO formatted timestamp of the data point.

September 3, 2025

4.10.5 Pages

AdminPage

isopruefi-frontend v1.0.0

isopruefi-frontend / pages/AdminPage

PAGES/ADMINPAGE

Functions

default

September 3, 2025



isopruefi-frontend v1.0.0

isopruefi-frontend / pages/AdminPage / default

Function: default()

default(): Element

Defined in: pages/AdminPage.tsx:18

 $\label{prop:prop:component} \mbox{ AdminPage component for managing locations, topics, and viewing weather data.}$

Provides controls for selecting location and temperature units, displays a weather chart, and includes management sections for locations and topics.

Also provides a logout button to clear authentication and redirect to sign-in.

Returns

Element

The rendered admin page.

September 3, 2025

UserPage

isopruefi-frontend v1.0.0

isopruefi-frontend / pages/UserPage

PAGES/USERPAGE

Functions

default

September 3, 2025

isopruefi-frontend v1.0.0

isopruefi-frontend / pages/UserPage / default

Function: default()

default(): Element

Defined in: pages/UserPage.tsx:15

UserPage component for viewing weather data and managing user preferences.

Provides controls for selecting location and temperature units, displays a weather chart, and includes a logout button to clear authentication and redirect to the welcome page.

Returns

Element

The rendered user page.

September 3, 2025

Welcome

isopruefi-frontend v1.0.0

isopruefi-frontend / pages/Welcome

PAGES/WELCOME

Functions

default

September 3, 2025

isopruefi-frontend v1.0.0

isopruefi-frontend / pages/Welcome / default

Function: default()

default(): Element

Defined in: pages/Welcome.tsx:27

Welcome page component that serves as the application's landing screen.

Features:

- Split-screen layout with logo and navigation
- Branded design with IsoPrüfi styling
- Navigation links to authentication pages
- Responsive design with centered content
- Consistent color scheme and typography

Returns

Element

JSX element containing the welcome page layout

Example

```
// Used in routing configuration
<Route path="/" element={<Welcome />} />
```

September 3, 2025

**

4.10.6 Utils

authApi

isopruefi-frontend v1.0.0

isopruefi-frontend / utils/authApi

UTILS/AUTHAPI

Type Aliases

• LoginResult

Functions

- login
- refreshToken
- register

September 3, 2025

isopruefi-frontend v1.0.0

```
isopruefi-frontend / utils/authApi / login
```

Function: login()

```
login( userName , password ): Promise \< LoginResult >
```

Defined in: utils/authApi.ts:69

Authenticates a user with username and password credentials.

Parameters userName

string

The user's login username

password

string

The user's password

Returns

Promise \< LoginResult >

Promise resolving to login tokens

Throws

When credentials are invalid or server error occurs

Example

```
try {
  const result = await login('user@example.com', 'password123');
  saveToken(result.token, result.refreshToken);
} catch (error) {
  console.error('Login failed:', error);
}
```

September 3, 2025

isopruefi-frontend / utils/authApi / refreshToken

Function: refreshToken()

```
refreshToken( token, refreshToken): Promise \< LoginResult >
```

Defined in: utils/authApi.ts:120

Refreshes an expired access token using a valid refresh token.

Parameters token

string

The expired JWT access token

refreshToken

string

The valid refresh token

Returns

```
Promise \< LoginResult >
```

Promise resolving to new authentication tokens

Throws

When refresh token is invalid, expired, or revoked

Example

```
try {
  const tokens = await refreshToken(oldToken, refreshToken);
  saveToken(tokens.token, tokens.refreshToken);
} catch (error) {
  // Refresh failed, redirect to login
  clearToken();
  window.location.href = '/login';
}
```

September 3, 2025

isopruefi-frontend / utils/authApi / register

Function: register()

```
register( userName , password ): Promise \< void >
```

Defined in: utils/authApi.ts:92

Registers a new user in the system. Requires admin privileges.

Parameters userName

string

The desired username for the new user

password

string

The password for the new user

Returns

Promise \< void >

Promise that resolves on successful registration

Throws

When registration fails, username exists, or insufficient permissions

Example

```
try {
   await register('newuser@example.com', 'securePassword123');
   console.log('User registered successfully');
} catch (error) {
   console.error('Registration failed:', error);
}
```

September 3, 2025

TYPE-ALIASES

isopruefi-frontend v1.0.0

isopruefi-frontend / utils/authApi / LoginResult

Type Alias: LoginResult

LoginResult = object

Defined in: utils/authApi.ts:12

Represents the result of a successful login operation.

Properties refreshToken

refreshToken: string

Defined in: utils/authApi.ts:16

Refresh token for obtaining new access tokens

token

token: string

Defined in: utils/authApi.ts:14

JWT access token for authenticated requests

September 3, 2025

config

isopruefi-frontend v1.0.0

isopruefi-frontend / utils/config

UTILS/CONFIG

Functions

• apiBase

September 3, 2025

isopruefi-frontend v1.0.0

isopruefi-frontend / utils/config / apiBase

Function: apiBase()

apiBase(): string

Defined in: utils/config.ts:22

Resolves the API base URL from runtime configuration or environment variables.

Priority order:

- 1. Runtime configuration from window.__APP_CONFIG__.API_BASE_URL
- 2. Build-time environment variable VITE_API_BASE_URL
- 3. Empty string as fallback

Returns

string

The API base URL with trailing slashes removed

Example

```
// Returns "https://api.example.com" (trailing slash removed)
const baseUrl = apiBase();
```

September 3, 2025

tokenHelpers

isopruefi-frontend v1.0.0

isopruefi-frontend / utils/tokenHelpers

UTILS/TOKENHELPERS

Interfaces

JwtPayload

Functions

- clearToken
- decodeToken
- getRefreshToken
- getToken
- getUserFromToken
- saveToken

September 3, 2025

<u>::</u>:

isopruefi-frontend v1.0.0

isopruefi-frontend / utils/tokenHelpers / clearToken

Function: clearToken()

clearToken(): void

Defined in: utils/tokenHelpers.ts:84

Removes both access and refresh tokens from localStorage.

Call this function when logging out or when tokens become invalid.

Returns

void

Example

```
// On logout
clearToken();
// Redirect to login page
```

September 3, 2025

isopruefi-frontend / utils/tokenHelpers / decodeToken

Function: decodeToken()

```
decodeToken(token): null | JwtPayload
```

Defined in: utils/tokenHelpers.ts:104

Decodes a JWT token and extracts the payload containing user information.

Does not verify the token signature - use only for reading claims.

Parameters token

string

The JWT token to decode

Returns

```
null | JwtPayload
```

The decoded payload object, or null if decoding fails

Example

```
const payload = decodeToken(accessToken);
if (payload) {
   console.log('Token expires at:', new Date(payload.exp * 1000));
}
```

September 3, 2025

 $is oprue fi-front end \ / \ utils/token Helpers \ / \ get Refresh Token$

Function: getRefreshToken()

```
getRefreshToken(): null | string
```

Defined in: utils/tokenHelpers.ts:69

Retrieves the stored refresh token from localStorage.

Returns

```
null | string
```

The refresh token string, or null if not found

Example

```
const refreshToken = getRefreshToken();
if (refreshToken) {
   // Use to obtain new access token
}
```

September 3, 2025

**

is oprue fi-frontend / utils/tokenHelpers / getToken

Function: getToken()

```
getToken(): null | string
```

Defined in: utils/tokenHelpers.ts:52

Retrieves the stored JWT access token from localStorage.

Returns

```
null | string
```

The access token string, or null if not found

Example

```
const token = getToken();
if (token) {
   // Use token for authenticated requests
}
```

September 3, 2025

**

is oprue fi-frontend / utils/tokenHelpers / getUserFromToken

Function: getUserFromToken()

getUserFromToken(token): null | string

Defined in: utils/tokenHelpers.ts:128

Extracts the user identifier from a JWT token.

The subject field typically contains the username or user ID.

Parameters token

string

The JWT token to extract user information from

Returns

null | string

The user identifier string, or null if extraction fails

Example

```
const currentUser = getUserFromToken(getToken());
if (currentUser) {
  console.log('Current user:', currentUser);
}
```

September 3, 2025

:

isopruefi-frontend / utils/tokenHelpers / saveToken

Function: saveToken()

saveToken(token, refreshToken): void

Defined in: utils/tokenHelpers.ts:34

Saves JWT tokens to browser local Storage for persistent authentication.

Parameters token

string

The JWT access token

refreshToken

string

The refresh token for obtaining new access tokens

Returns

void

Example

```
// After successful login saveToken(response.accessToken, response.refreshToken);
```

September 3, 2025

•

INTERFACES

*

isopruefi-frontend v1.0.0

```
isopruefi-frontend / utils/tokenHelpers / JwtPayload
 Interface: JwtPayload
  Defined in: utils/tokenHelpers.ts:10
  Standard JWT payload structure with common claims.
  Extends to allow additional custom claims from the authentication system.
Indexable
  [key:string]:unknown
  Allow additional custom claims
Properties exp?
  optional exp: number
  Defined in: utils/tokenHelpers.ts:14
  Expiration time (Unix timestamp)
iat?
     optional iat: number
  Defined in: utils/tokenHelpers.ts:16
  Issued at time (Unix timestamp)
sub?
    optional sub: string
  Defined in: utils/tokenHelpers.ts:12
  Subject (usually user ID or username)
September 3, 2025
```

5. Docker

5.0.1 Docker Environment Documentation

This documentation provides a comprehensive overview of the Docker environments used in the IsoPruefi project, including development and live deployments, container configurations, and environment file management.

Development Environment

The development environment is configured via <code>docker-compose.yml</code> in the project root and provides local development with automatic certificate generation for localhost domains.

CONTAINER OVERVIEW (DEVELOPMENT)

Container	Image	Description	Access
traefik	traefik:3.4.4	Reverse proxy and load balancer with automatic HTTPS certificates	traefik.localhost, Ports: 80 , 443 , Dashboard: 8432
influxdb3	influxdb:3.2.1-core	Time series database (InfluxDB 3.x) for sensor and telemetry data	Port: 8181
influxdb-explorer	<pre>influxdata/influxdb3- ui:1.0.3</pre>	Web interface for managing and querying InfluxDB data	explorer.localhost, Port: 8888
postgres	postgres:alpine3.21	PostgreSQL database for relational data storage	Port: 5432
loki	grafana/loki:3.5.2	Log aggregation and storage system	Port: 3100
prometheus	prom/prometheus:v3.4.2	Monitoring system for metrics collection and alerting	Port: 9090
alloy	grafana/alloy:v1.9.2	Observability data collector for Loki and Prometheus integration	Ports: 12345 , 4317 , 4318
grafana	grafana/grafana:12.0.2	Visualization and dashboarding for metrics and logs	grafana.localhost
isopruefi-frontend-1	Built from ./isopruefi- frontend	React frontend application (development build)	frontend.localhost
isopruefi-backend- api-1	Built from ./isopruefi- backend	.NET REST API for application logic	backend.localhost
isopruefi-mqtt- receiver-1	Built from ./isopruefi- backend	MQTT message receiver and processor	Internal service (no external access)
isopruefi-get-weather- worker-1	Built from ./isopruefi- backend	Weather data collection worker service	Internal service (no external access)

Live/Production Environment

The live environment is configured via <code>isopruefi-docker-live/docker-compose.yml</code> and uses pre-built Docker images from GitHub Container Registry with load balancing and health checks.

CONTAINER OVERVIEW (LIVE)

Container	Image	Description	Access
traefik	traefik:3.4.4	Production reverse proxy with custom domain routing	Ports: 5000 (HTTP), 5001 (HTTPS), 5002 (Dashboard)
influxdb3	influxdb:3.2.1-core	Production time series database with persistent volumes	Port: 5006
postgres	postgres:17.5	Production PostgreSQL with persistent volumes	Port: 5003
loki	grafana/loki:3.5.2	Production log aggregation with persistent storage	Internal service
prometheus	prom/prometheus:v3.4.2	Production metrics collection with persistent storage	Port: 5004
alloy	grafana/alloy:v1.9.2	Production observability data collector	Internal service
grafana	grafana/grafana:12.0.2	Production dashboards with persistent configuration	Port: 5005
isopruefi-frontend-1/2	<pre>ghcr.io/deadmade/isopruefi- frontend:latest</pre>	Load-balanced React frontend instances	Port: 5007 (frontend-1 only), Path: /frontend
isopruefi-backend- api-1/2	<pre>ghcr.io/deadmade/isopruefi- rest-api:latest</pre>	Load-balanced .NET API instances	Path: /backend (via Traefik)
isopruefi-mqtt- receiver-1/2	<pre>ghcr.io/deadmade/isopruefi- mqtt-worker:latest</pre>	Redundant MQTT receiver instances	Internal services
isopruefi-get-weather- worker-1/2	<pre>ghcr.io/deadmade/isopruefi- weather-worker:latest</pre>	Redundant weather data collection instances	Internal services

LIVE ENVIRONMENT URLS

When deployed on the production server (${\tt aicon.dhbw-heidenheim.de}$), the services are accessible via:

- Traefik Dashboard: https://aicon.dhbw-heidenheim.de:5002
- Frontend Application: https://aicon.dhbw-heidenheim.de:5001/frontend
- Backend API: https://aicon.dhbw-heidenheim.de:5001/backend
- Grafana Dashboards: http://aicon.dhbw-heidenheim.de:5005
- Prometheus Metrics: http://aicon.dhbw-heidenheim.de:5004
- PostgreSQL: aicon.dhbw-heidenheim.de:5003
- InfluxDB: http://aicon.dhbw-heidenheim.de:5006

Note: HTTPS is available on port 5001, HTTP on port 5000. The frontend and backend use path-based routing with Traefik handling SSL termination.

Networks

DEVELOPMENT NETWORKS

- isopruefi-network: Main network for application services and Traefik
- isopruefi-monitoring: Dedicated network for observability tools (Loki, Prometheus, Grafana, Alloy)

LIVE NETWORKS

- · isopruefi-network: Main production network for all services
- isopruefi-monitoring: Production observability network

Key Differences: Development vs Live

DEVELOPMENT ENVIRONMENT

- · Build Strategy: Local builds from source code with hot-reload capabilities
- Domains: Uses *.localhost domains with automatic certificate generation
- · Scaling: Single instance of each service
- Data: Bind mounts for development data persistence
- Environment: ASPNETCORE_ENVIRONMENT=Development

LIVE ENVIRONMENT

- · Build Strategy: Pre-built images from GitHub Container Registry
- Domains: Custom domain routing with production certificates
- · Scaling: Multiple instances with load balancing for high availability
- Data: Named Docker volumes for production data persistence
- · Health Checks: Comprehensive health monitoring for all services
- Environment: ASPNETCORE_ENVIRONMENT=Docker
- Backup: Automated backup system (currently commented out)

Service Details

TRAEFIK CONFIGURATION

- Development: Automatic HTTPS with local certificates for *.localhost domains
- · Live: Production routing with custom domain support and SSL termination

DATABASE SERVICES

- InfluxDB: Time-series data storage for sensor readings and telemetry
- PostgreSQL: Relational data for user management, configuration, and application state

OBSERVABILITY STACK

- Loki: Centralized logging with automatic log collection via Docker labels
- Prometheus: Metrics collection and alerting
- · Grafana: Unified dashboards for logs and metrics visualization
- · Alloy: Data collection and forwarding to Loki/Prometheus

APPLICATION SERVICES

- Frontend: React-based web application for data visualization and control
- Backend API: .NET REST API providing core application logic
- MQTT Receiver: Handles incoming sensor data via MQTT protocol
- Weather Worker: Collects external weather data from multiple APIs

September 2, 2025

deadmade, maratin23

5.1 Environment Files

5.1.1 Overview

The IsoPruefi project uses environment files to manage configuration across different deployment environments:

- Development Environment: secrets.env (project root)
- Live Environment: isopruefi-docker-live/secrets.env
- Backup Configuration: isopruefi-docker-live/backup/*.env

5.1.2 Main Environment Files (secrets.env)

Development Environment (./secrets.env)

Used by the development Docker Compose setup for local development with localhost domains.

```
==== Database Configuration =====
# PostgreSQL Database Settings
POSTGRES_HOST=postgres
POSTGRES_PORT=5432
POSTGRES DATABASE=Isopruefi
POSTGRES_USERNAME=Isopruefi
POSTGRES_PASSWORD=secret
# Admin user for initial system setup
Admin__UserName=admin
Admin__Email=admin@localhost.dev
Admin__Password=DevAdmin123!
# Connection String (uses above variables)
ConnectionStrings__DefaultConnection=Host=${POSTGRES_HOST};Port=${POSTGRES_PORT};Database=${POSTGRES_DATABASE};Username=${POSTGRES_USERNAME};Password=${POSTGR
ES PASSWORD}
# ==== Authentication ====
# JWT Configuration for API authentication
Jwt__ValidIssuer=localhost
Jwt__ValidAudience=localhost
Jwt__Secret=your-secure-256-bit-secret-key-replace-this-in-production
     == Time-Series Database =====
# InfluxDB Settings for time-series data storage
Influx__InfluxDBHost=http://influxdb3:8181
Influx__InfluxDBToken=
Influx__InfluxDBDatabase=IsoPruefi
   ==== MQTT Configuration =====
# MQTT Broker settings for IoT sensor communication
Mqtt__BrokerHost=aicon.dhbw-heidenheim.de
Mqtt__BrokerPort=1883
# ==== Weather API Configuration ====
# External weather service URLs and location
Weather__OpenMeteoApiUrl=https://api.open-meteo.com/v1/forecast?latitude=48.678&longitude=10.1516&models=icon_seamless&current=temperature_2m
Weather__BrightSkyApiUrl=https://api.brightsky.dev/current_weather?lat=48.67&lon=10.1516
Weather__Location=Heidenheim
VITE_API_BASE_URL=https://backend.localhost
```

Live Environment (isopruefi-docker-live/secrets.env)

Used by the production Docker Compose setup with additional production-specific configurations.

```
# ===== Database Configuration =====
# PostgreSQL Database Settings
POSTGRES_HOST=postgres
POSTGRES_PORT=5432
POSTGRES_DB=1sopruefi  # Note: Different variable name for production
POSTGRES_USER=1sopruefi  # Note: Different variable name for production
POSTGRES_USER=Suspressuspressuspressure
# Admin user for initial system setup
Admin__UserName=admin
Admin__Email=admin@localhost.dev
Admin__Password=DevAdmin123!
```

```
# Connection String (uses above variables)
ConnectionStrings_DefaultConnection=Host=${POSTGRES_HOST};Port=${POSTGRES_PORT};Database=${POSTGRES_DB};Username=${POSTGRES_USER};Password=${POSTGRES_PASSWORD}
# ==== Authentication =====
\ensuremath{\text{\# JWT}} Configuration for API authentication
Jwt__ValidIssuer=localhost
Jwt__ValidAudience=localhost
\verb|Jwt_Secret=your-secure-256-bit-secret-key-replace-this-in-production|\\
 # ===== Time-Series Database =====
# InfluxDB Settings for time-series data storage
Influx_InfluxDBHost=http://influxdb3:8181
Influx__InfluxDBToken=
Influx__InfluxDBDatabase=IsoPruefi
# ===== MQTT Configuration =====
# MQTT Broker settings for IoT sensor communication
 Mqtt__BrokerHost=aicon.dhbw-heidenheim.de
Mqtt__BrokerPort=1883
# ===== Weather API Configuration =====
# External weather service URLs and location
Weather__OpenMeteoApiUrl=https://api.open-meteo.com/v1/forecast?latitude=48.678&longitude=10.1516&models=icon_seamless&current=temperature_2m
We ather\_BrightSkyApiUrl=https://api.brightsky.dev/current\_weather?lat=48.67\&lon=10.1516\\ We ather\_NominatimApiUrl=https://nominatim.openstreetmap.org/search?format=jsonv2\&postalcode=Weather\_Location=Heidenheim
# ===== Grafana Configuration =====
# Grafana admin password for production access GF_SECURITY_ADMIN_PASSWORD=
```

5.1.3 Configuration Variables Reference

Database Configuration

Variable	Description	Dev Value	Live Value	Required
POSTGRES_HOST	PostgreSQL hostname	postgres	postgres	~
POSTGRES_PORT	PostgreSQL port	5432	5432	~
POSTGRES_DATABASE / POSTGRES_DB	Database name	Isopruefi	Isopruefi	V
POSTGRES_USERNAME / POSTGRES_USER	Database username	Isopruefi	Isopruefi	V
POSTGRES_PASSWORD	Database password	secret	secret	V

Authentication & Security

Variable	Description	Example Value	Required
AdminUserName	Initial admin username	admin	V
AdminEmail	Initial admin email	admin@localhost.dev	V
AdminPassword	Initial admin password	DevAdmin123!	V
JwtValidIssuer	JWT token issuer	localhost	V
JwtValidAudience	JWT token audience	localhost	V
JwtSecret	JWT signing secret (256-bit)	your-secure-256-bit-secret-key	V

InfluxDB Configuration

Variable	Description	Example Value	Required
InfluxInfluxDBHost	InfluxDB connection URL	http://influxdb3:8181	V
InfluxInfluxDBToken	InfluxDB authentication token	apiv3	💢 (Dev), 🔽 (Live)
InfluxInfluxDBDatabase	InfluxDB database name	IsoPruefi	V

MQTT Configuration

Variable	Description	Example Value	Required
MqttBrokerHost	MQTT broker hostname	aicon.dhbw-heidenheim.de	V
MqttBrokerPort	MQTT broker port	1883	V

Weather API Configuration

Variable	Description	Purpose	Required
WeatherOpenMeteoApiUrl	Open-Meteo API endpoint with coordinates	Primary weather data source	✓
WeatherBrightSkyApiUrl	BrightSky API endpoint with coordinates	Secondary weather data source	V
WeatherNominatimApiUrl	OpenStreetMap geocoding service	Location lookup (Live only)	×
WeatherLocation	Human-readable location name	Display purposes	V

Frontend Configuration

Variable	Description	Example Value	Required
VITE_API_BASE_URL	Backend API base URL for frontend	https://backend.localhost	V

Grafana Configuration (Live Only)

Variable	Description	Example Value	Required	
GF_SECURITY_ADMIN_PASSWORD	Grafana admin user password	1234	(Live)	

September 2, 2025

deadmade

6. System Documentation

6.1 Database Schema

6.1.1 Overview

IsoPrüfi uses a dual database approach:

- PostgreSQL with Entity Framework Core for relational data (users, settings, coordinates)
- InfluxDB for time-series sensor data and metrics

6.1.2 Core Entities

ApiUser

Extends ASP.NET Core Identity for user authentication.

CoordinateMapping

Stores geographic coordinates for postal codes with usage tracking.

Column	Туре	Description
PostalCode (PK)	int	Unique postal code identifier
Location	string	Location name
Latitude	double	Geographic latitude
Longitude	double	Geographic longitude
LastUsed	DateTime?	When postal code was last updated
LockedUntil	DateTime?	Temporary locked to get the current Temperature of the Location

TopicSetting

 $\ensuremath{\mathsf{MQTT}}$ topic configuration with sensor metadata.

Column	Туре	Description
TopicSettingId (PK)	int	Auto-generated identifier
CoordinateMappingId (FK)	int	Links to CoordinateMapping
DefaultTopicPath	string	MQTT topic prefix (max 100 chars)
GroupId	int	Group identifier
SensorTypeEnum	SensorType	Type of sensor
SensorName	string	Sensor name (max 50 chars)
SensorLocation	string	Sensor location (max 50 chars)
HasRecovery	bool	Recovery feature enabled

6.1.3 Enumerations

SensorType

Available sensor types:

- temp Temperature
- spl Sound pressure level
- hum Humidity
- ikea IKEA sensor
- co2 CO2 sensor
- mic Microphone

6.1.4 Relationships

- TopicSetting \rightarrow CoordinateMapping (Many-to-One)
- · ApiUser uses ASP.NET Identity tables

6.1.5 Migration Commands

```
cd isopruefi-backend dotnet ef migrations add <MigrationName> --project ./Database/Database.csproj --startup-project ./Rest-API/Rest-API.csproj
```

6.1.6 InfluxDB Schema

Measurements

TEMPERATURE

Sensor temperature readings from Arduino devices.

Field/Tag	Туре	Description
value (field)	float	Temperature measurement
sensor (tag)	string	Sensor identifier
sequence (tag)	int	Message sequence number
timestamp	DateTime	Measurement timestamp

OUTSIDE_TEMPERATURE

External weather data from APIs.

Field/Tag	Туре	Description
value (field)	float	Temperature in Celsius
value_fahrenheit (field)	float	Temperature in Fahrenheit
postalcode (field)	int	Associated postal code
place (tag)	string	Location name
website (tag)	string	Data source API
timestamp	DateTime	Measurement timestamp

UPTIME

Arduino device availability tracking.

Field/Tag	Туре	Description
sensor (field)	string	Sensor identifier
timestamp	DateTime	Uptime check timestamp

September 3, 2025

DianaTin23, deadmade

6.2 MQTT Protocol Documentation

6.2.1 Overview

IsoPrüfi uses MQTT for sensor data communication between Arduino devices and the backend system. The protocol handles real-time data transmission with recovery capabilities for offline scenarios.

6.2.2 Topic Structure

Standard Topic Format

```
{topicPrefix}/{sensorType}//{sensorId}
```

Example: dhbw/ai/si2023/temp/Sensor_One

Topic Components

- topicPrefix: Configurable prefix (default: dhbw/ai/si2023/)
- sensorType: Type of sensor (temp, hum, co2, spl, mic, ikea)
- sensorId: Unique sensor identifier

6.2.3 Message Formats

Standard Sensor Reading

```
{
    "timestamp": 1672531200,
    "value": [23.5],
    "sequence": 1234
}
```

Recovery Data (Bulk Upload)

```
{
    "timestamp": 1672531200,
    "value": [23.5],
    "sequence": 1234,
    "meta": {
        "t": [1672531100, 1672531120],
        "v": [23.1, 23.3, 23.4],
        "s": [1231, 1232, 1233]
    }
}
```

6.2.4 Data Fields

Field	Туре	Description
timestamp	DateTime	Unix timestamp (seconds since epoch)
value	float[]	Sensor readings (array for multi-point sensors)
sequence	int	Sequential counter for message ordering
meta.t	DateTime[]	Recovery timestamps
meta.v	float[]	Recovery values
meta.s	int[]	Recovery sequence numbers

6.2.5 Error Handling

- Failed publishes trigger local storage
- Recovery data sent in batches (max 3 files per loop)
- 60-second timeout for recovery operations
- Automatic reconnection on connection loss

September 3, 2025

DianaTin23, deadmade

6.3 Monitoring & Observability

6.3.1 Overview

IsoPrüfi uses Grafana, Prometheus, and Loki for comprehensive monitoring and observability.

6.3.2 Components

Grafana Dashboards

• Location: /grafana/

- · Features:
- System metrics visualization
- · Application performance monitoring
- · Custom uptime dashboard
- Alerting

Prometheus Metrics

• Config: /loki/prometheus.yml

· Scrapes: Application health endpoints

• Alerts: Defined in alerts.yml

Loki Logging

• Config: /loki/loki-config.yaml

• Agent: Grafana Alloy for log collection (config.alloy)

· Centralized: All application and system logs

6.3.3 Health Checks

Monitoring Targets

- Application uptime
- Database connectivity
- MQTT broker status
- · Worker process health

6.3.4 Alerting

Configure alerts in /loki/alerts.yml:

- Service down alerts
- High error rate notifications
- Resource utilization warnings

6.3.5 Log Analysis

Access logs through Grafana's Explore feature:

- Filter by service/component
- Search for error patterns
- Trace request flows

September 3, 2025

deadmade

7. IsoPrüfi Documentation

7.1 Introduction and Goals

7.1.1 Aim of our project IsoPrüfi:

Our project aims to test the effectiveness of building insulation based on outside temperature and present the data clearly using diagrams.

7.1.2 Features

Must-Have

- A website for a user-friendly presentation of temperature comparison diagrams
- Reliable sensors that measure interior temperature
- The ability to retrieve outside temperature data
- · Clusterization of containers that we create ourselves

Should-Have

- Sensors should be capable of storing temperature data for a period of one day, even in the absence of an internet connection or synchronization with the server
- A website should be used to offer configuration options

Could-Have

· Database clustering

Won't Have

- The containers will only run on one server, however they are designed to function independently of each other
- · Since this is a software project, we won't implement any resilience on the hardware side

7.1.3 Requirements Overview

Functional Requirements

- The system must provide three data sources: two for indoor measurements and one for outdoor measurements
- Data should be updated every 60 seconds
- Each data point must include both temperature and timestamp
- Users must be able to view diagrams and evaluations of the collected data
- · Users should have access to historical data to observe long-term trends
- The system must use containers for deployment
- In case of no network and or MQTT broker connection, the temperature data will be saved on an SD card for up to 24 h

Non-Functional Requirements

- The system should achieve an availability of 99.5%
- ${\mbox{\ensuremath{\bullet}}}$ The system must remain reliable even if one container fails
- Data must be persistently stored in the database
- Automated unit tests must cover core functionalities, including correct data transmission, successful data storage, and simulation of failure scenarios

7.1.4 Quality Goals

Quality Goal	Description
Persistence	Sensor readings must be logged centrally (database) and locally (SD card), if offline -> No data loss
Data Integrity	Data must include timestamps and sequance to prevent corruption or duplication
Availability	The system must remain partially operational during network outages and recover automatically

7.1.5 Stakeholders

Role/Name	Expectations	Influence
Developer	Solution that is easy to maintain and fulfills all requirements for the project	Quality of Code, Clean Architecture, Final product
Supervisor	Correct methodology, clear documentation and tracability of results	Sets expectations and reviews the final product
Coaches	Clear documentation, preparation of meetings and clear presentation of the results for each meeting	Review of the final product and support for the implementation
User/Owner	Want to reduce their heating costs through stable temperature measurements and correct assessment of the building's isolation	Requires easy usability and trustworthy temperature data
Systemadministrator	Stable infrastructure, easy deployments and clear logs for easy maintenance	Configuration of the system

August 31, 2025

⇔DianaTin23, deadmade, maratin23

7.2 Quality Requirements

7.2.1 1. Persistence

Temperature data must be reliably and permanently stored in the database, even in the event of temporary system or connection failures.

Measurable Criteria:

- Data Loss Rate: A maximum of 0.1% of all recorded measurements may be lost.
- Successful Write Operations: At least 99.9% of all database write operations must be completed without error.
- Fallback Storage: In case of missing connectivity, temperature data is written to the local SD card for up to 24h and synchronized once the connection is restored.
- · Retry and Confirmation: Failed write operations to the central database are retried until confirmation is received.

Testability:

- Disconnect the system from the internet in a controlled way and verify that data is buffered on the SD card and later persisted in the database.
- · Simulate database outages to check retry logic and final persistence.
- Run long-term operation tests with daily storage cycles (e.g., multiple days) to verify absence of data loss.
- Use SQL queries to compare the expected number of measurements with the actual count in the database, and to calculate the percentage of successful write operations, ensuring compliance with the defined thresholds.

7.2.2 2. Data Integrity

The recorded data must be correct, complete, and plausible to enable a reliable evaluation of the building's insulation.

Measurable Criteria:

- Inconsistent Data Rate: Less than 0.05% of all records may be duplicates, incorrect, or implausible.
- · Validation Error Rate: A maximum of 0.1% of data may be rejected by validation mechanisms.
- · Automatic plausibility checks:
 - Range validation: Outdoor readings must stay between -30 °C and 45 °C, indoor readings between -10 °C and 35 °C. Values outside this range are logged as warnings.
 - Jump detection: Sudden jumps >10 °C between consecutive readings are flagged.
 - Difference and mean analysis: Consecutive differences and moving averages are tracked to detect anomalies.
 - Statistical window checks: Mean and standard deviation over a defined time window are used to identify abnormal fluctuations.

Testability:

- · Inject out-of-range or implausible test data and verify that the system logs warnings or rejects values.
- · Simulate sudden temperature jumps to ensure they are flagged.
- · Compare sensor readings against expected ranges (indoor vs. outdoor).

7.2.3 3. Availability

The system must remain functional even in the event of partial failures, so that users can always access the temperature data. Each critical service is deployed redundantly with at least two instances. If one instance fails, Traefik automatically routes traffic to the backup instance. All containers expose health checks, and stateless design ensures fast restart and recovery.

The system is resilient against the following single-instance failures:

- Website (frontend): one instance down → second instance continues serving requests
- REST API: one instance down → second instance handles API traffic
- Weather Data Worker: one instance down → second instance continues scheduled tasks
- MQTT Receiver: one instance down → second instance continues message processing

Measurable Criteria:

- System Availability: ≥ 99.5% overall operational time (software side)
- Frontend Data Availability: ≥ 99.5% of the time, current or last available data is accessible via the UI
- · Resilience Mechanisms:
 - Redundant service instances per cluster (frontend, backend, workers)
 - Traefik load balancer distributes traffic and enables failover
 - Stateless service design for automatic restart or replacement
 - · Health checks for all major containers
 - · Local SD storage at Arduino nodes ensures sensor data buffering during backend or network outages

Testability:

- Controlled shutdown of one instance per cluster (frontend, REST API, Weather Data Worker, MQTT Receiver) to verify automatic failover via Traefik
- Disable one database or monitoring component to confirm health checks and recovery strategies
- · Simulate network outage between Arduino and backend to verify SD-card buffering and later synchronization
- Long-term monitoring of uptime metrics to confirm compliance with ≥ 99.5% availability

September 1, 2025

DianaTin23

7.3 Architecture Constraints and Solution Strategy

7.3.1 Architecture Contraints

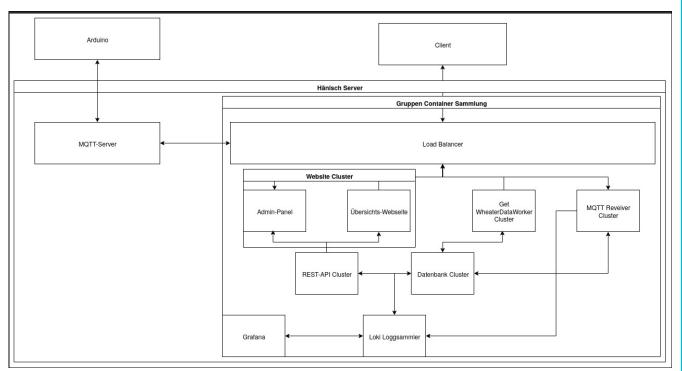
Category	Constraint
Technical	The project will be hosted on a single server provided by Prof. Hänisch
Technical	Indoor temperature measurement hardware is supplied by the university
Technical	At least two data sources are required, with at least one being an Arduino device
Technical	The hardware and database are not specifically designed for high reliability
Technical	The final system must run in a clean environment with no prior setup required
Organizational	Weekly meetings with a coach are scheduled for project discussions
Political	The submission deadline is the 04.09.2025

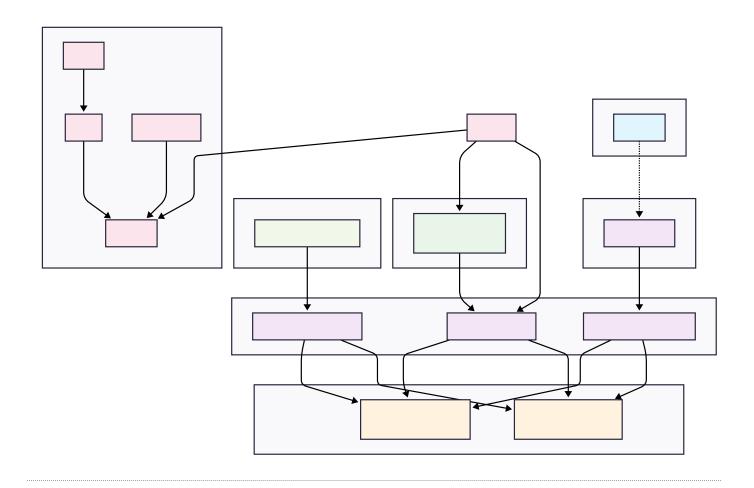
7.3.2 System Decomposition Strategy

All services are containerized and grouped under a shared container orchestration layer. The system is fronted by a load balancer, ensuring scalability and high availability. MQTT acts as the bridge between the hardware (Arduino sensors) and the backend. Logging and observability are handled via the Loki stack, and external visibility is offered via a user-facing website and Grafana deshboards.



All services are containerized and grouped under a shared container orchestration layer (labelled "Gruppen Container Sammlung"). The system is fronted by a load balancer, ensuring scalability and high availability. MQTT acts as the bridge between the hardware (Arduino sensors) and the backend. Logging and observability are handled via the Loki stack, and external visibility is offered via a user-facing website and Grafana deshboards.





7.3.3 Organizational / Development Process Decisions

- Source Control: GitHub with structured branches and CI pipelines
- Documentation: Based on arc42 template, managed in MkDocs
- Infrastructure as Code: All services defined in docker-compose.yml and version-controlled
- Architecture Decisions: Documented using ADRs (Architecture Decision Records). The detailed technology choices are documented in 04 Architecture Decisions

7.3.4 Cross-cutting Concepts

The following concepts ensure consistency and support the quality goals:

Domain Concepts

- $\bullet \Delta T \ metric: \ Temperature \ difference \ between \ indoor \ and \ outdoor \ sensors \ as \ key \ indicator.$
- Sensor units: Each Arduino node has a unique ID and physical placement (e.g., north vs. south façade).
- Enriched records: Each reading contains value, timestamp, and source ID.

Fault Tolerance

- Redundant logging: Default to central database, with SD card fallback on network/MQTT outages (buffering up to 24h).
- MQTT QoS (1): Guarantees at-least-once delivery.
- Stateless services: Enable fast restart and failover without data inconsistencies.

Architecture and Design Patterns

- Message-driven architecture: Asynchronous data flow via MQTT.
- Microservices: Independent services for ingestion, enrichment, API.
- API gateway pattern: REST API shields internal complexity and exposes a single entry point.

Development Concepts

- Containerization: Consistent deployment via Docker.
- Continuous Integration: Automated checks and tests before merges.
- Infrastructure as Code: Networks and dependencies tracked in source control.

Operational Concepts

- Observability: Logs via Loki, metrics via Prometheus, dashboards via Grafana.
- Health monitoring: /health endpoints with automated restart on failure.
- Scalability: Additional instances (frontend, MQTT receivers, workers) can be added behind Traefik.

August 31, 2025

DianaTin23, deadmade

7.4 Architecture Decisions

7.4.1 ADR 1: Backend Technology Stack

Status:

Accepted (July 2025)

Context:

System needs robust backend technology for REST API and worker services. Team has existing familiarity with C# development.

Decision:

Use .NET 9 with C# for all backend services (REST API, MQTT Receiver, Weather Worker).

Alternatives Considered:

Option	Pros	Cons
Node.js	Rapid iteration, huge ecosystem	Different stack; weaker static typing by default; less team experience
Go	High perf/concurrency; small binaries	Less team experience; different tooling
Python	Rich libs; fast prototyping	Lower throughput; weaker typing by default

Consequences:

Positive:

- Team familiarity reduces development time
- Strong typing prevents runtime errors
- Excellent tooling and debugging support
- $\bullet \ \mathsf{Modern} \ \mathsf{async/await} \ \mathsf{support} \ \mathsf{for} \ \mathsf{I/O} \ \mathsf{operations}$

Negative:

- Platform dependency (though mitigated by containers)
- Larger memory footprint than some alternatives

Neutral:

· Containerization standardizes runtime

7.4.2 ADR 2: Microservices Architecture

Status:

Accepted (July 2025)

Context:

System has distinct responsibilities: API serving, MQTT message processing, and weather data fetching. Need modularity and independent scaling.

Decision:

Split backend into separate services: REST API, MQTT Receiver Worker, Weather Data Worker.

Alternatives Considered:

Option	Pros	Cons
Monolith	Simple deploy; easy local dev	No independent scaling; fault blast radius
Modular monolith	Clear boundaries in one process	Still one deploy unit; limited isolation
Serverless	No servers to manage; auto-scale	Cold starts; platform coupling; ops visibility variance

Consequences:

Positive:

- Clear separation of concerns
- Independent scaling and deployment
- · Fault isolation between services

Negative:

- Increased deployment complexity
- Network communication overhead between services

Neutral:

• Requires basic observability to manage complexity

7.4.3 ADR 3: Dual Database Strategy

Status:

Accepted (July 2025)

Context:

System needs both structured application data (users, authentication) and time-series sensor data with different access patterns.

Decision:

Use PostgreSQL for application data and InfluxDB for time-series sensor data.

Alternatives Considered:

Option	Pros	Cons
PostgreSQL + TimescaleDB	One stack; SQL everywhere	Ops complexity; perf tuning for time series needed
InfluxDB only	Optimized for time series	Awkward relational modeling; joins missing
SQLite + InfluxDB Lite	Simple, lightweight	Limited concurrency; feature gaps

Consequences:

Positive:

- PostgreSQL optimized for relational data and transactions
- \bullet InfluxDB optimized for time-series queries and compression
- Each database serves its specific use case efficiently

Negative:

• Two databases to maintain and backup

Neutral:

• Extract, Transform, Load (ETL) between stores is minimal

7.4.4 ADR 4: Observability Stack

Status:

Accepted (July 2025)

Context:

Distributed microservices architecture requires comprehensive monitoring, logging, and alerting capabilities.

Decision:

Loki for logs, Prometheus for metrics, Grafana for dashboards, Alloy as agent.

Alternatives Considered:

Option	Pros	Cons
ELK (Elasticsearch, Kibana)	Powerful search/analytics	Heavier footprint; more ops effort
OTel collector + vendor	Standards-based; flexible pipelines	Vendor lock-in and/or cost
Managed cloud observability	Minimal ops	Ongoing costs; data residency limits

Consequences:

Positive:

- Complete observability into system health and performance
- $\bullet \ \text{Industry-standard tools with good integration} \\$
- Unified dashboard for all monitoring data

Negative:

· Additional infrastructure to maintain

Neutral:

• Can swap components later

7.4.5 ADR 5: Traefik as Reverse Proxy

Status:

Accepted (July 2025)

Context:

Multiple services need unified entry point, SSL termination, and service discovery in containerized environment.

Decision:

Use Traefik as reverse proxy with automatic service discovery and HTTPS termination.

Alternatives Considered:

Option	Pros	Cons
Nginx	Mature; high performance	Manual routing/config; no auto-discovery
Caddy	Simple TLS; easy config	Fewer discovery features
HAProxy	Very fast; robust LB features	More manual config; fewer HTTP niceties

Consequences:

Positive:

- Automatic service discovery via Docker labels
- Built-in SSL certificate management
- · Load balancing capabilities

Negative:

- · Single point of failure if not properly configured
- · Additional configuration complexity

Neutral:

• Replaceable by Nginx if needed

7.4.6 ADR 6: JWT Authentication Strategy

Status:

Accepted (July 2025)

Context:

REST API requires secure authentication mechanism. Need stateless authentication for microservices architecture.

Decision:

Implement JWT token-based authentication with Entity Framework for user management.

Alternatives Considered:

Option	Pros	Cons
Server-side sessions	Simple; revocation is trivial	Stateful; sticky sessions; scale limits
OAuth2/OIDC proxy	Standards-based; SSO ready	More moving parts; infra complexity
API keys	Simple; easy for machines	Poor granularity; rotation burdens

Consequences:

Positive:

- · Stateless authentication scales well
- Standard approach with good library support
- Tokens can carry user claims

Negative:

- Token revocation complexity
- Requires secure token storage on client side

Neutral:

• Token TTL balances risk and UX

7.4.7 ADR 7: Docker Compose for Development Environment

Status:

Accepted (July 2025)

Context:

Complex multi-service architecture needs consistent development environment setup across team members.

Decision:

Use Docker Compose to orchestrate all services for local development.

Alternatives Considered:

Option	Pros	Cons
Dev Containers	Great DX; reproducible	Editor-coupled; learning curve
Kind/Minikube	Closer to k8s	Heavier locally; slower feedback
Scripts/Makefiles	Minimal tooling	Fragile; drift across machines

Consequences:

Positive:

- Consistent development environment
- Easy service dependency management
- Simplified onboarding for new developers

Negative:

- Requires Docker knowledge from all developers
- Resource intensive on development machines

Neutral:

• Can migrate to Kubernetes later

7.4.8 ADR 8: Frontend

Status:

Accepted (July 2025)

Context:

System needs a frontend to display charts from measured/collected temperature data and to generate API docs with TypeDoc.

Decision:



v0

JavaScript React app via Docker. Reason: quick start.

Issue: Schema changes not caught at build time caused runtime UI errors (no static typing).



٧1

TypeScript React with Create React App (CRA). Reason: typing and better tooling. Issue: TypeDoc generation failed due to CRA/tooling version conflicts.

React + TypeScript built with Vite for the frontend.

Alternatives Considered:

Option	Pros	Cons
CRA (TS)	Familiar, out-of-the-box setup	Tooling conflicts with TypeDoc
Next.js	SSR/ISR, ecosystem	Unneeded complexity for our use
Custom Webpack	Full control	More maintenance

Consequences:

Positive:

- TypeDoc works
- faster startup
- · lean tooling

Negative:

· Some devs must learn Vite

Neutral:

· No server-side rendering (SSR) required

7.4.9 ADR 9: Hardware Platform Decision (board, sensors)

Status:

Accepted (July 2025)

Context:

MKR1010 and ADT7410 were provided. Requirements: offline buffering, precise time, dual sites.

Decision:

Use Arduino MKR1010 with RTC DS3231 and SD card; deploy two identical units.

Alternatives Considered:

Option	Pros	Cons
ESP32 boards	Wi-Fi integrated; strong community	Different toolchain; requalification
Different sensors	Potential accuracy/cost benefits	Revalidation effort; integration risk
Single hardware unit	Simpler setup	No north/south comparison; less robust

Consequences:

Positive:

- Local data persistence via SD card enables offline data storage for ≤24h
- Timestamp reliability through RTC with battery
- Compact hardware, low power, WiFi-ready (MKR1010)

Negative:

- RTC and SD modules require additional wiring and SPI/I2C handling
- Time must be synchronized manually once (e.g., via compile-time setting or initial sync)

Neutral:

- The Arduino MKR1010 was predefined, not evaluated
- · Final visualization and backend will depend on further platform choices (e.g., MQTT, REST, database)

7.4.10 ADR 10: Arduino Development Environment Decision

Status:

Accepted (July 2025)

Context:

Arduino firmware needs modular builds and host-side unit tests.

Decision:

PlatformIO for builds; Unity with native target for tests.

Alternatives Considered:

Option	Pros	Cons
Arduino IDE	Easy; official	No native tests; inflexible structure
CMake toolchain	Flexible; IDE-agnostic	More setup; custom plumbing
Ceedling	Solid C test framework	Extra integration effort

Consequences:

Positive:

- Reproducible builds and consistent project structure
- PC-native unit tests for business logic (Unity, native target)
- Seamless VS Code integration
- Use of modern C++ structure and dependency management

Negative:

- Additional setup effort for non-Arduino users (e.g., Unity, test runners)
- Developers must learn PlatformIO's structure (src/lib/test)

Neutral:

- The PlatformIO toolchain abstracts away the underlying GCC setup
- Unit tests cannot cover board-specific behavior (e.g., Wire, SD, RTC) directly without mocks

7.4.11 Sources

Documenting Architecture Decisions by Michael Nygard

August 31, 2025

DianaTin23, deadmade

7.5 Context and Scope

7.5.1 Technical Context

The IsoPrüfi system operates in a distributed container-based architecture hosted on the DHBW Server infrastructure. It integrates multiple services for data ingestion, processing, storage, and visualization.

Components and Channels

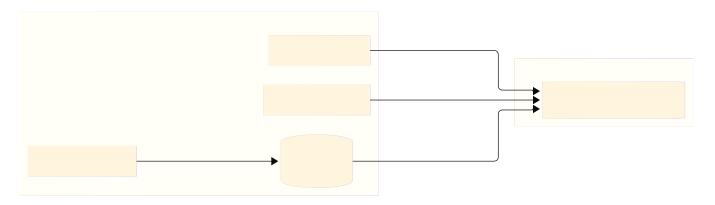
Component	Communication Channel	Description
Arduino	Wi-Fi / MQTT	Publishes temperature readings to the MQTT broker
MQTT Broker	MQTT	External broker for sensor communication
traefik	HTTP / HTTPS	Reverse proxy and load balancer, entry point to the system
isopruefi-backend	HTTP (REST)	Provides unified access to system data
isopruefi-frontend	HTTPS	User interface for visualizing measurements
influxdb	TCP / SQL	Time-series database for sensor data
postgres	TCP / SQL	Relational database for application data
loki	gRPC / HTTP	Collects and stores logs from all services
prometheus	НТТР	Collects metrics from services
grafana	НТТР	Dashboard for metrics and logs
alloy	HTTP / gRPC	Integrates Loki and Prometheus for observability
Weather API	HTTPS	Provides external weather data
Client (Browser)	HTTPS	Interacts with the frontend

For a detailed list of all Docker containers, their images, addresses and networks, see the separate documentation page: Docker Development Environment

Mapping I/O to Channels

I/O Type	Channel	Source	Destination
Temperature Reading	WiFi / MQTT	Arduino	MQTT Broker → MQTT Receiver Worker
Weather Data Pull	HTTPS (API)	Weather API Service	Weather Data Worker
Web Page Access	HTTPS	Client Browser	Traefik → React Frontend
API Request	HTTP / REST	Frontend (via Traefik)	REST API Service
Data Storage	SQL / TCP	Backend Services	PostgreSQL (app data), InfluxDB (time series)
System Logs	gRPC / HTTP	All Services	Alloy \rightarrow Loki/Prometheus (visualized in Grafana)

Technical Context Diagram



August 31, 2025

DianaTin23, deadmade

7.6 Risks and Technical Debts

7.6.1 1. Description of the Process/System

Overview of the Entire Product:

- Temperature Measurement and Transmission:
 - Measuring temperature via temp sensors (and RTC modules) and APIs as a source for outside weather data
 - Transmission of data via WiFi/MQTT from the Arduino to the MQTT Broker
 - · Saving the data on the Arduino in case of loss of connection
- · Data Storage:
 - Databases: Postgres and Influx
 - Caching of data on local storage in case of problems
- · Analysis/Evaluation:
 - · Results are displayed on a frontend web page

Components Involved:

- · Temperature Measurement:
 - Temperature sensors, RTC modules, Arduino and SD module/card for inside temperatures
 - Online weather via API for outside temperature
- · Temperature Transmission:
 - · Network/WiFi
- · Data Storage:
 - Database systems: Postgres and Influx
 - Caching
- · Visualization/Analysis:
 - Data availability, website, analytics platforms

Process Aspects:

- Data flow throughout the system
- · Handling of failure and recovery scenarios

7.6.2 2. Error Analysis

Possible Errors

- Incorrect or missing data
- Unavailability of services or functions (e.g., website, Grafana)

Causes

- Compatibility issues due to software or hardware updates
- · Security vulnerabilities

· Temperature Measurement:

- Sensor errors (e.g., incorrect calibration, hardware malfunction, sensor failure, power supply issues, incorrect interval configuration)
- Misassignment of data (e.g., north/south confusion)
- Weather service outages
- Outage of containers processing data

· Data Transmission:

- Network outages or connectivity issues
- Duplicate data transmission

· Data Storage:

- Incorrect or duplicate entries
- Database corruption or failure

Visualization/Analysis:

- · Website unavailability
- Incorrect data presented for visualization

Impacts

- Gaps in data analysis
- · Misinterpretation or incorrect assessment of results
- Lack of long-term evaluation or comparison basis
- · No or limited access to collected data

7.6.3 3. Evaluation of Errors and Consequences

Error	Probability of Occurrence	Severity	Probability of Detection	Risk Priority Number
Sensor error	2-3 (unlikely)	8-9 (severe)	2-3 (inevitable detection)	32-81
Misassignment of data	3 (low)	6-7 (disturbance)	5-6 (only detected during targeted checks)	90-126
Weather service outage	1 (almost impossible)	8-9 (severe)	2-3 (inevitable detection)	16-27
Network outage	2 (unlikely)	8-9 (severe)	2-3 (inevitable detection)	31-45
Duplicate transmission	2 (unlikely)	2 (irrelevant)	5-6 (only detected during targeted checks)	20-24
Incorrect/missing entries	2-3 (unlikely)	8-9 (severe)	3-4 (high probability of detection)	48-108
Database corruption	2-3 (unlikely)	8-9 (severe)	3-4 (high probability of detection)	48-108
Website/Grafana malfunction	1 (almost impossible)	8-9 (severe)	2-3 (inevitable detection)	16-27
Power outage	3 (low)	8-9 (severe)	2-3 (inevitable detection)	48-81

Error	Probability of Occurrence	Severity	Probability of Detection	Risk Priority Number
Sensor error (wrong temp data)	2-3 (unlikely)	8-9 (severe)	4-5 (high probability of detection)	64-135
Sensor unavailability	2-3 (unlikely)	8-9 (severe)	2-3 (inevitable detection)	32-81
Misassignment of data	3 (low)	6-7 (disturbance)	5-6 (only detected during targeted checks)	90-126
Weather service outage	1 (almost impossible)	7-8 (severe)	2-3 (inevitable detection)	16-27
Network outage	8 (likely)	3-4 (irrelevant)	3-4 (high probability of detection)	72-128
Duplicate transmission	2 (unlikely)	2 (irrelevant)	5-6 (only detected during targeted checks)	20-24
Incorrect/missing entries	2-3 (unlikely)	6-7 (disturbance)	3-4 (high probability of detection)	36-84
Database corruption	2-3 (unlikely)	8-9 (severe)	3-4 (high probability of detection)	48-108
Database unavailability	2-3 (unlikely)	3-4 (irrelevant)	3-4 (high probability of detection)	18-48
Website malfunction	2-3 (unlikely)	8-9 (severe)	2-3 (inevitable detection)	32-81
Power outage	3 (low)	8-9 (severe)	2-3 (inevitable detection)	48-81

7.6.4 4. Corrective actions

	Diele Drierite Nember	Mitigation Massure
Error	Risk Priority Number	Mitigation Measure
Sensor error	32-135	-
Misassignment of data	90-126	Implement data validation and labeling checks
Weather service outage	16-27	Use fallback data sources
Network outage	31-45	Local storage of data on the Arduino
Duplicate transmission	20-24	-
Incorrect/missing entries	48-108	Input validation
Database corruption	48-108	-
Website malfunction	16-27	Monitor uptime

Risk Priority Number	Mitigation Measure
32-81	In case of missing measurements -> alerting
90-126	Implement data validation
16-27	Use fallback data source
31-45	Local storage of data on the Arduino
20-24	-
48-108	Input validation
48-108	-
16-27	Monitor uptime
18-81	Clustering of containers, monitoring uptime and caching of data
	32-81 90-126 16-27 31-45 20-24 48-108 48-108 16-27

7.6.5 5. Technical Debts

Debt	Impact	Mitigation	Priority
Single-server deployment (no HA for DB/Traefik)	Outage stops whole system; RTO/RPO undefined	Define RTO/RPO; periodic restore drills; consider DB replication later	High
External single MQTT broker	Ingestion is SPOF; no controlled failover	Document broker SLA; add reconnect/ backoff; plan broker redundancy/bridge later	High
SD-card buffering deduplication	Risk of duplicate inserts on reconnect	Idempotent writes (sensorId + timestamp + seq unique); DB upsert/unique index	High
Time synchronisation of sensors	Clock drift \rightarrow wrong ΔT and ordering	Regular NTP sync or backend time anchor; RTC drift check procedure	High
Missing/uneven health/ readiness endpoints	Load balancer may route to bad pods	Standardize /health and /ready; Traefik forward-auth or ping checks	Medium
No alerting rules/SLOs	Failures unnoticed; 99.5% not enforced	Prometheus alert rules + Grafana alerts; SLO dashboards for availability	Medium
Secrets in env files	Leakage risk; no rotation	Use Docker secrets; rotate regularly; restrict file perms; avoid committing	High
TLS/auth on MQTT not specified	Data spoofing/sniffing possible	Enable TLS; client auth (user/pass or certs); topic ACLs	High
Schema/migration strategy	Breaking changes risk data loss	Versioned EF migrations; InfluxDB bucket retention + downsampling plan	Medium
Config scattering (topics, URLs)	Drift and hidden coupling	Central config per env; validated at startup; document defaults	Medium
Limited automated fault tests	Availability regressions unnoticed	CI: chaos/failure tests (DB down, broker down, network flap)	Medium
Weather API limits/caching	Rate-limit failures; latency	Add caching, retries with jitter, circuit breaker, fallback to last-known	Low
Backup without periodic restore test	False sense of safety	Quarterly restore test; document runbook; verify integrity checks	High
Logging/PII retention not defined	Storage bloat; compliance risk	Retention policy in Loki; scrub PII; log level guidelines	Medium
Rate limiting/DoS on API	Resource exhaustion	Traefik rate limits; API quotas; request size limits	Medium
Ownership/runbooks	Slow incident response	Define service owners; on-call matrix; SOPs for common incidents	Low

7.6.6 Sources

FMEA from the Orgahandbuch (Bundesministerium des Inneren)

September 3, 2025

⇔DianaTin23, deadmade, maratin23

8. License

arc42

Parts of this documentation are based on the arc42 architecture template, licensed under Creative Commons Attribution 4.0 International (CC BY 4.0). © Dr. Gernot Starke, Dr. Peter Hruschka et al.

SJuly 14, 2025

♣ DianaTin23