



Siemens
Industry
Online
Support

APPLICATION EXAMPLE

Creating User-Defined Web Pages with the Web API for SIMATIC S7-1200 G2 / S7-1500

SIMATIC STEP 7 (TIA PORTAL V20) / SIMATIC S7-1200 G2 / SIMATIC S7-1500

SIEMENS

Legal information

Use of application examples

Application examples illustrate the solution of automation tasks through an interaction of several components in the form of text, graphics and/or software modules. The application examples are a free service by Siemens AG and/or a subsidiary of Siemens AG ("Siemens"). They are non-binding and make no claim to completeness or functionality regarding configuration and equipment. The application examples merely offer help with typical tasks; they do not constitute customer-specific solutions. **The application examples are not subject to standard tests and quality inspections of a chargeable product and may contain functional and performance defects or other faults and security vulnerabilities. You are responsible for the proper and safe operation of the products in accordance with all applicable regulations, including checking and customizing the application example for your system, and ensuring that only trained personnel use it in a way that prevents property damage or injury to persons. You are solely responsible for any productive use.**

Siemens grants you the non-exclusive, non-sublicensable and non-transferable right to have the application examples used by technically trained personnel. Any change to the application examples is your responsibility. Sharing the application examples with third parties or copying the application examples or excerpts thereof is permitted only in combination with your own products. Any further use of the application examples is explicitly not permitted and further rights are not granted. You are not allowed to use application examples in any other way, including, without limitation, for any direct or indirect training or enhancements of AI models.

Disclaimer of liability

Siemens shall not assume any liability, for any legal reason whatsoever, including, without limitation, liability for the usability, availability, completeness and freedom from defects of the application examples as well as for related information, configuration and performance data and any damage caused thereby. This shall not apply in cases of mandatory liability, for example under the German Product Liability Act, or in cases of intent, gross negligence, or culpable loss of life, bodily injury or damage to health, non-compliance with a guarantee, fraudulent non-disclosure of a defect, or culpable breach of material contractual obligations. Claims for damages arising from a breach of material contractual obligations shall however be limited to the foreseeable damage typical of the type of agreement, unless liability arises from intent or gross negligence or is based on loss of life, bodily injury or damage to health. The foregoing provisions do not imply any change in the burden of proof to your detriment. You shall indemnify Siemens against existing or future claims of third parties in this connection except where Siemens is mandatorily liable.

By using the application examples you acknowledge that Siemens cannot be held liable for any damage beyond the liability provisions described.

Other information

Siemens reserves the right to make changes to the application examples at any time without notice and to terminate your use of the application examples at any time. In case of discrepancies between the suggestions in the application examples and other Siemens publications such as catalogs, the content of the other documentation shall have precedence.

The Siemens terms of use (<https://www.siemens.com/global/en/general/terms-of-use.html>) shall also apply.

Cybersecurity information

Siemens provides products and solutions with industrial cybersecurity functions that support the secure operation of plants, systems, machines and networks.

In order to protect plants, systems, machines and networks against cyber threats, it is necessary to implement – and continuously maintain – a holistic, state-of-the-art industrial cybersecurity concept. Siemens' products and solutions constitute one element of such a concept.

Customers are responsible for preventing unauthorized access to their plants, systems, machines and networks. Such systems, machines and components should only be connected to an enterprise network or the internet if and to the extent such a connection is necessary and only when appropriate security measures (e.g. firewalls and/or network segmentation) are in place.

For additional information on industrial cybersecurity measures that may be implemented, please visit www.siemens.com/cybersecurity-industry.

Siemens' products and solutions undergo continuous development to make them more secure. Siemens strongly recommends that product updates are applied as soon as they are available and that the latest product versions are used. Use of product versions that are no longer supported, and failure to apply the latest updates may increase customer's exposure to cyber threats.

To stay informed about product updates, subscribe to the Siemens Industrial Cybersecurity RSS Feed under <https://www.siemens.com/cert>.

Table of contents

1.	Preface.....	5
2.	Overview of the System.....	6
2.1.	User-defined Web Applications with the S7 Web API	8
3.	Application Example	9
3.1.	Hardware and Software Components Used	9
3.2.	Installation of the Application Example	11
3.2.1.	Procedure for Creating a Web Page	11
3.2.2.	Configuring the Hardware	12
3.2.3.	Download of the S7 User Program onto the S7-CPU	13
4.	Operating the Application Example	19
4.1.	Operating the web page "Tank Overview"	20
4.2.	Operating the web page "Data View"	24
4.3.	Operating the web page "Diagnostic Buffer"	25
4.4.	Operating the web page "Syslog Buffer"	26
4.5.	Operating the web page "Alarms"	27
5.	Function Mechanisms of this Application Example	28
5.1.	Functional Principle of the S7 User Program	28
5.1.1.	Startup (OB100)	29
5.1.2.	Main (OB1)	29
5.1.3.	TankSimu (FB1)	30
5.2.	Functionality of the HTML Files with Typescript	30
5.2.1.	Typescript-SIMATIC-s7-webserver-api.....	30
5.2.2.	Angular	31
5.2.3.	Style sheet: Bootstrap	32
5.2.4.	Webpack in Angular projects	32
5.3.	Web Application Structure	33
5.3.1.	Main Configuration	33
5.3.1.1.	Typescript configuration.....	33
5.3.1.2.	Angular main configuration.....	34
5.3.1.3.	Custom webpack configuration	34
5.3.1.4.	Build tools configuration	35
5.3.1.5.	Linters	35
5.3.1.6.	Base files	35
5.3.1.7.	Application Entry and Global Resources.....	36
5.3.1.8.	Assets.....	37
5.3.2.	Angular Application	37

5.3.2.1.	Components	37
5.3.2.2.	Services	40
5.3.2.3.	Configuration	45
5.3.2.4.	Routing	47
5.3.2.5.	Angular Development Server	48
6.	Appendix	49
6.1.	Service and support	49
6.2.	Links and literature	50
6.3.	Change documentation	51

1. Preface

Aim of the application example

This application example illustrates the possibility of creating "user-defined web applications" on the SIMATIC S7-1200 G2 / S7-1500 by using the S7 Web API and standard HTML editors. It also explains the process of downloading these pages into the S7 PLC by using the Web Application Manager.

Main contents of the application example

This application example highlights the following key points:

- Fundamentals for user-defined web applications
- Creating user-defined web applications with S7 Web API
- Configuring the web server for using user-defined web applications
- Download of a Web Application Project into the PLC

NOTE

Users of this manual should have a certain level of web development expertise to fully understand the content. Developing a web application with the S7 Web API demands advanced web development expertise, particularly proficiency in JavaScript and HTML.

The application example and the web server should not and cannot replace an HMI system.

2. Overview of the System

Web Server and System Pages in the S7 CPU

Modern automation technology is evolving to integrate internet technologies alongside Ethernet-based communication, which enables direct access to systems through the intranet using standards like HTTP and HTTPS. This integration allows for flexible evaluation, diagnostics, and control of controllers over large distances, utilizing mobile devices such as tablets and smartphones.

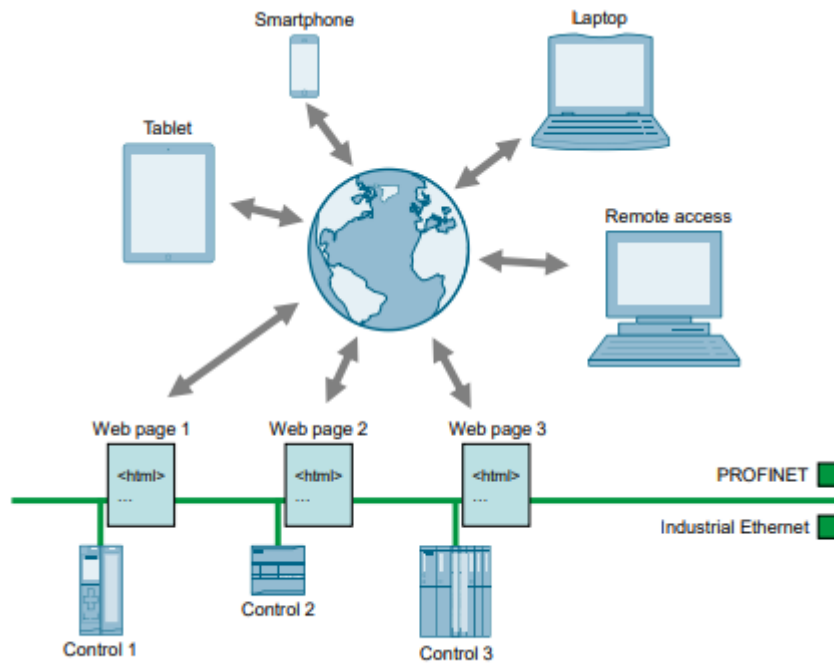


Figure 2-1: Overview of the Automation System and Web Server Architecture

During the test and commissioning phase, engineers require flexible CPU access to visualize individual data during operations for diagnostic purposes. The advantage of accessing control data through various web browsers is that it can be displayed and controlled to a limited degree on any computer or web-enabled device without the need for additional software.

The S7 CPU web server offers standard web pages, including identification, diagnostic buffer, module status, alarms, communication, topology, and file browser. Additionally, users have the option to create custom web applications tailored to their specific use cases and requirements. This manual is dedicated to guiding you through the creation of user-defined web applications using the S7 Web API.

Advantages of the Web Server

- **Integrated web pages.** The standard web pages for easy viewing of service and diagnostics information are activated with one click. Additionally, individually created, user-defined web pages can be generated.
- **Location-independent.** Web pages can be accessed worldwide via a standard internet browser. Access is possible via mobile communication devices, e.g., tablet PC, smartphone.
- **Application example.** Universal use of the application example for all control systems.
- Reduced working hours through simple enabling of the web server.
- Time saving in planning and implementing your automation solution through simple adjustment and deployment of the web application example.

User-defined Web Applications using the S7 Web API

Currently, there are two technologies available for developing custom web applications to connect with the S7 CPU:

1. Automation Web Programming (AWP) – Requires STEP 7 compilation.
2. S7 Web API – can be developed and tested independently from the PLC code.

This document specifically focuses on the development of custom web pages using the Web API and web applications. To create your user-defined web page with S7 Web API, you can use tools such as Visual Studio Code or Notepad++. For designing your web page, you can use all possibilities provided to you by HTML, CSS (Cascading Style Sheets), and JavaScript.

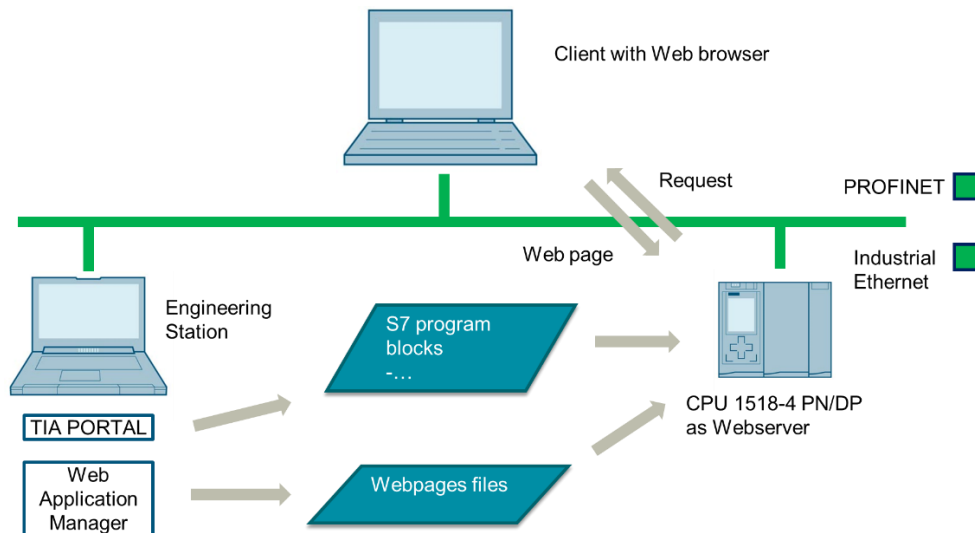


Figure 2-2: Overview of the overall situation

2.1. User-defined Web Applications with the S7 Web API

The development of user defined web applications using the S7 Web API is entirely independent of S7 CPU program development. This fact allows for modifications and testing of the web application without impacting the S7 CPU user program.

Thanks to the Web API and the Web Application Manager, the creation of a web application and its transfer is completely independent from the development of the S7 CPU program in the TIA Portal. Once the location of the process data in the S7 CPU is defined (DB), a web developer can start development.

The following steps describe the procedure for creating user-defined web pages:

PLC Program Development

1. Use SIMATIC STEP 7 (TIA Portal) to develop the PLC program, ensuring it includes program code, hardware configuration, and Web Server activation.
2. Download all program blocks onto the CPU to ensure proper implementation.

Web Page Development

1. Create the HTML file for the user-defined web page and link it to the PLC using the S7 Web API.

Web Project Download onto the PLC

1. Use the Web Application Manager to download the web project to the CPU.

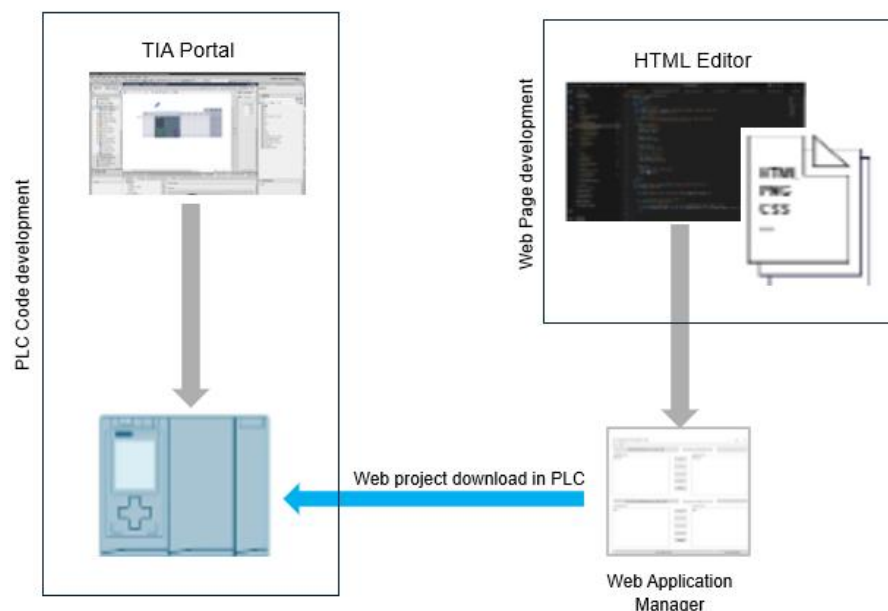


Figure 2-3: Procedure of creating user-defined web pages

3. Application Example

This application example illustrates the development of a custom web project incorporating a front-end framework in Angular. It presents the S7 Web API methods necessary for accessing S7 CPU's tags and parameters. The implementation combines HTML with Web API, using TypeScript/JavaScript in conjunction with Angular.

3.1. Hardware and Software Components Used

This application example was implemented with a CPU1518-4 PN/DP and a CPU 1214C DC/DC/DC.

A computer serves for the creation of the PLC project and the web content, as well as for loading them to the CPU and displaying the web pages in a web browser.

The application example was created with the following components:

Hardware Components

Component	Item number	Note
CPU 1518 - 4 PN/DP	6ES7 518-4AP00-0AB0	FW V3.1
CPU 1214C DC/DC/DC	6ES7 214-1AH50-0XB0	FW V1.0

Table 3-1

Software Components

Component	Item number	Note
SIMATIC STEP 7 Professional V20	6ES7822-1AE24-0YA5	-
Software for creating HTML files and scripts, e.g., Visual Studio Code		-
SIMATIC Step 7 Typescript Webserver API	GitHub - siemens/typescript-simatic-s7-webserver-api	Library used to facilitate communication between S7 CPU and user-defined web pages
Web Application manager	simatic-s7-webserver-api/samples/WebAppManager at main · siemens/simatic-s7-webserver-api · GitHub	Tool used to download the Web application onto the S7 CPU
Web browser, e.g., Google Chrome, Microsoft Edge		-

Table 3-2

The following web browsers were explicitly tested for communication with the CPUs:

- Google Chrome
- Microsoft Edge

Content of the application example

You will find the following in-depth content in the application example:

- Configuration of the web server for a CPU with PN interface
- Creation of a user-defined web page for the CPU with the following functions:
 - Displaying CPU tags.
 - Graphic display of CPU tags.
 - Setting of CPU tags.
 - Displaying texts which are linked with CPU tags.
 - Display of images that are linked to tags of the CPU.
 - Going to web pages with links in the navigation bar (Angular management).
 - Display of Diagnostics Buffer of the CPU.
 - Display of Syslog Buffer of the CPU.
 - Alarms display with acknowledgment function implemented.
 - Cyclic update of the web pages with Angular script.
 - Login form to access to CPU.
- Particularities in the S7 user program creation:
 - Providing tags for the web page.
 - Further processing of tags from the web page in the S7 user program.

Code of the Application Example

Components	Comments
68011496_CreatingUserDefinedWebPages_CODE_V50.zip	The zip file contains the STEP 7 project with the corresponding scripts and html files needed in the web application. The web application files are in an independent folder from the STEP 7 project called myApp located in another folder called webPages. It is included also another demo example to try out with a web browser called ApiDemoPage
68011496_CreatingUserDefinedWebPages_DOC_V50_en.pdf	This document

Table 3-3

3.2. Installation of the Application Example

Hardware installation

The computer with the web browser must be connected to the CPU via Industrial Ethernet. There are multiple ways to do it, but the simplest solution is to directly connect it with one of the PN interfaces of the CPU.

NOTE

Please observe the installation and connection guidelines from the corresponding manuals.

Installing the software

1. Install SIMATIC STEP 7 (TIA Portal).
2. Install a web page creation tool on the computer that you want to create the web page. It is recommended to have an IDE (integrated development environment) to facilitate the use of third-party software packages and other benefits like the developer tools, integrated terminal, compiler. For example, Visual Studio Code.
3. Install a web browser such as Google Chrome or Microsoft Edge on the computer with which you want to access the web page on the CPU.
4. Install the Web Application Manager on the CPU to download the web application to the PLC.

3.2.1. Procedure for Creating a Web Page

The configuration and settings in STEP 7 and the writing of the HTML file are closely related. The following procedure is recommendable for that:

1. Configuring the Hardware and the Web Server settings.
2. Creating the program blocks and tags in the S7 user program.
3. Compile and download the S7 user program.

Thanks to the Web API and the Web Application Manager, the creation of the web application and its transfer is completely independent from the development of the S7 user program. Once it is defined where the data is going to be located (definition of the data blocks), the developer can start the process:

4. Creating the HTML Files.
5. Download the web application with the Web Application Manager

3.2.2. Configuring the Hardware

1. Start the TIA Portal and select "Project > New..." to create a new project with the name "Webserver_S7_1500" or "Webserver_S7_1200", for example.
2. Add a S7-1500 or S7-1200 G2 station via "Add new device > Controller > SIMATIC S7-1500/ SIMATIC S7-1200 G2". The Device View of the CPU opens.
3. Assign the IP address 192.168.0.1 of the CPU to PROFINET Interface X1. Via this IP address, you will later use the web browser to access the web page loaded on the CPU.

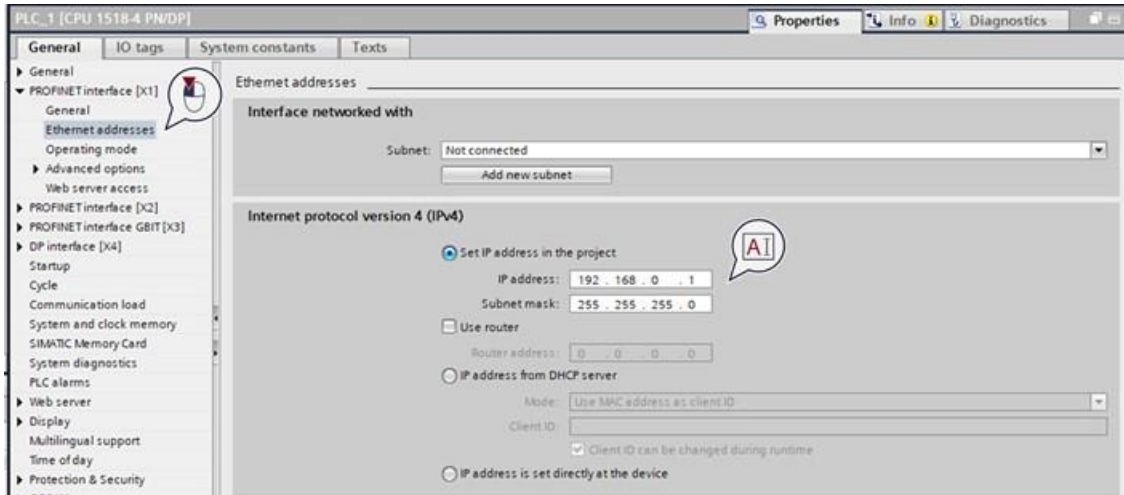


Figure 3-1: PLC Configuration

4. Click the "Web Server Access" option in the properties of the PROFINET Interface. Enable web server via IP address of this interface.

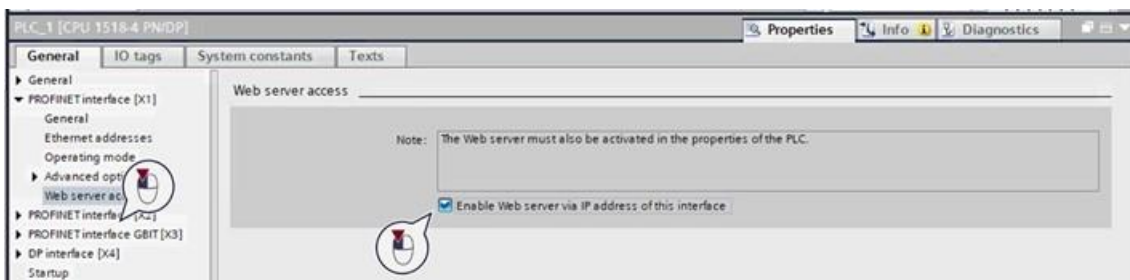


Figure 3-2: PLC Configuration, web server access

5. Click on "Web server" under the General configuration of the PLC and select activate web server on this module.

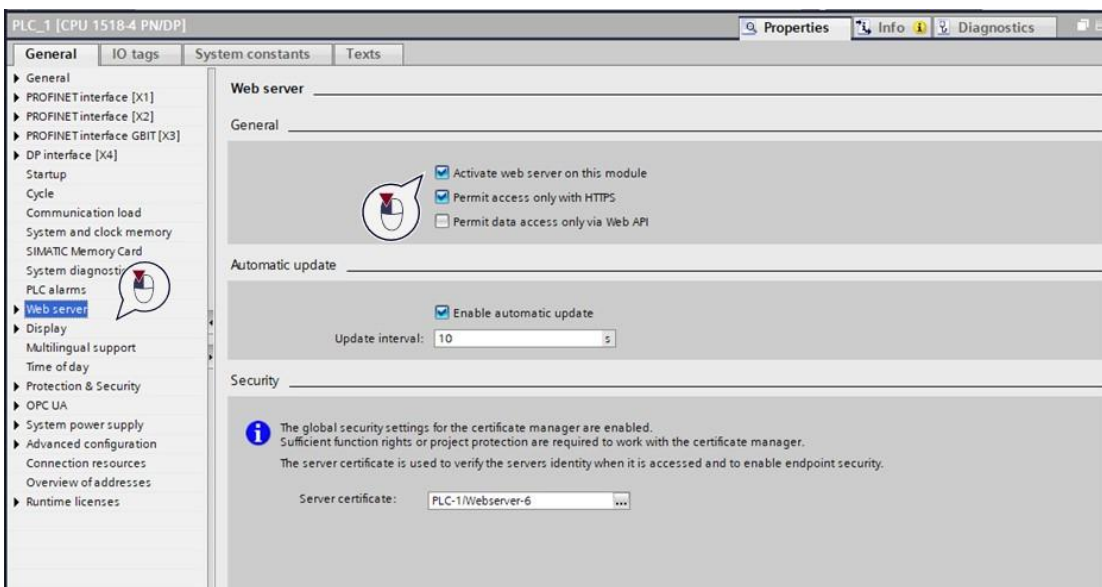


Figure 3-3: PLC configuration, web server configuration

6. On the project tree configuration select: Security Settings > Settings > Project Protection. Click on "Protect this project". This will enable some security features like for example the certificate manager where you can download the CA certificate and device certificate. Once a project is protected it can't be unprotected anymore.

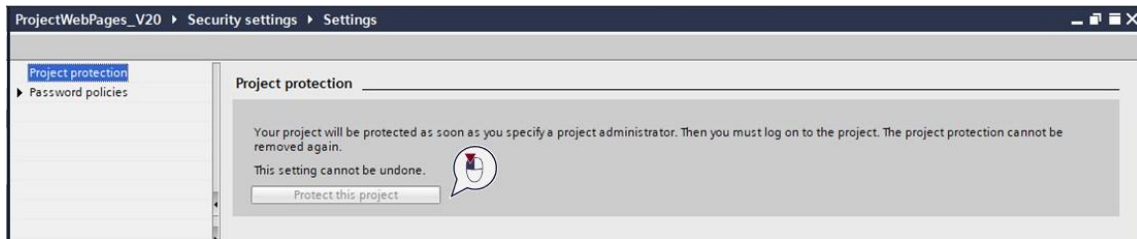


Figure 3-4: Project configuration, protection

7. Enter the following login data and confirm with "OK":
 - a. Username: UserAdministrator
 - b. Password: Siemens123!
8. Go to Users and roles and there you will find the previously created user. Now create a new user with the following data:
 - a. Username: User
 - b. Password: Siemens123!

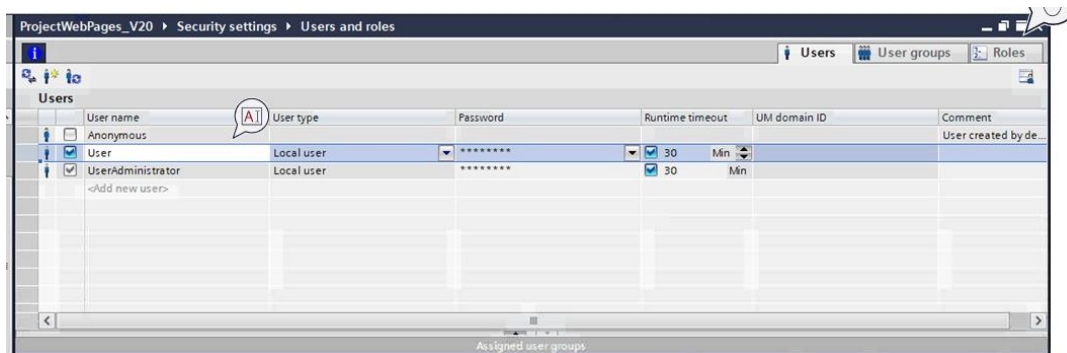


Figure 3-5: PLC Configuration, users and roles

9. The new role has the function rights for each PLC of the project: S7-1500 and S7-1200 G2 described at the end of this chapter.

3.2.3. Download of the S7 User Program onto the S7-CPU

To proceed with the installation of the Application Example you need to have SIMATIC STEP7 (TIA Portal) and a SIMATIC S7-1500 or S7-1200 G2 CPU¹⁾ or optional PLCSIM Advanced installed in your computer.

To put the application example into operation, proceed as follows:

1. Unzip the file "68011496_CreatingUserDefinedWebPages_CODE_V50.zip"
2. Start SIMATIC STEP 7 (TIA Portal V20) and open the project into TIA Portal. Verify in SIMATIC STEP 7 (TIA Portal) that the following settings of the CPU are correct:
3. The web server must be activated.
4. The Web API needs the HTTPS transmission protocol, so to run this application example this protocol will be necessary.
5. Start PLCSIM Advanced or connect a real device.
6. Switch to the Device View in TIA Portal
7. Select the SIMATIC S7-1500 or the S7-1200 G2 CPU¹⁾ and download the code.

NOTE

¹⁾ SIMATIC S7-1200 G2 does not work with PLCSIM Advanced

Download of the Web Application Example onto the S7-CPU

To download the application example into the PLC you need to start the Web Application Manager. This application is an open-source application that allows a user to simply perform the download of a complete web application to the PLC, making the process completely independent from TIA Portal.



Reference to repository: [simatic-s7-webserver-api/samples/WebAppManager at main · siemens/simatic-s7-webserver-api · GitHub](https://github.com/siemens/simatic-s7-webserver-api/samples/WebAppManager)

Here you can download an executable that contains all the necessary files to run the application

How to download an application to the PLC

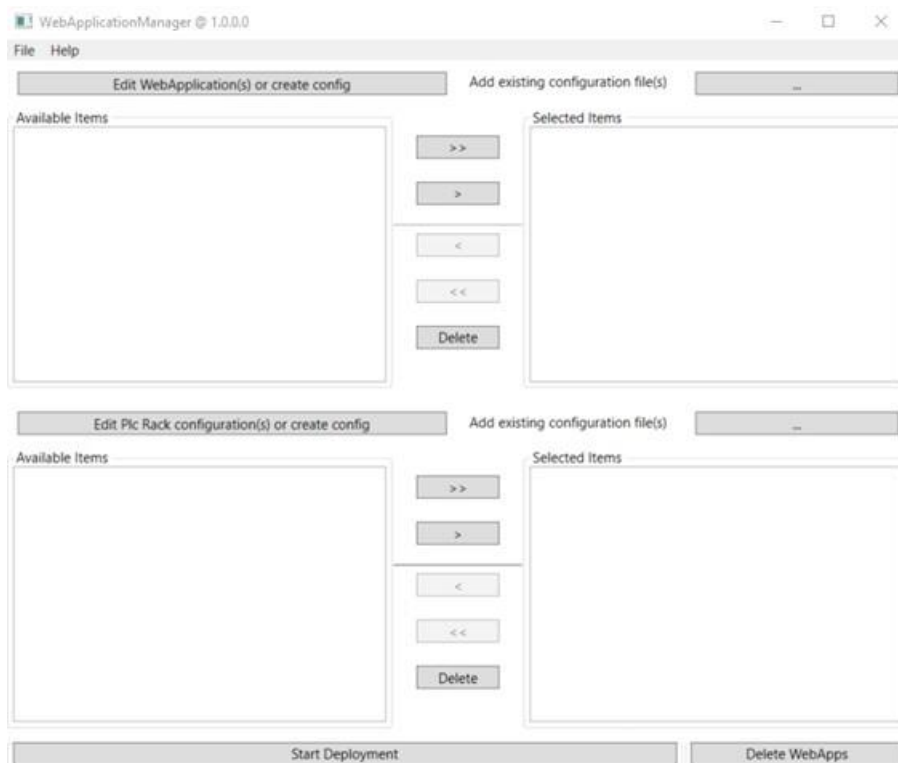


Figure 3-6: Web Application Manager interface

1. Edit Web Application(s) or create config: Create a configuration for the web application to be downloaded.

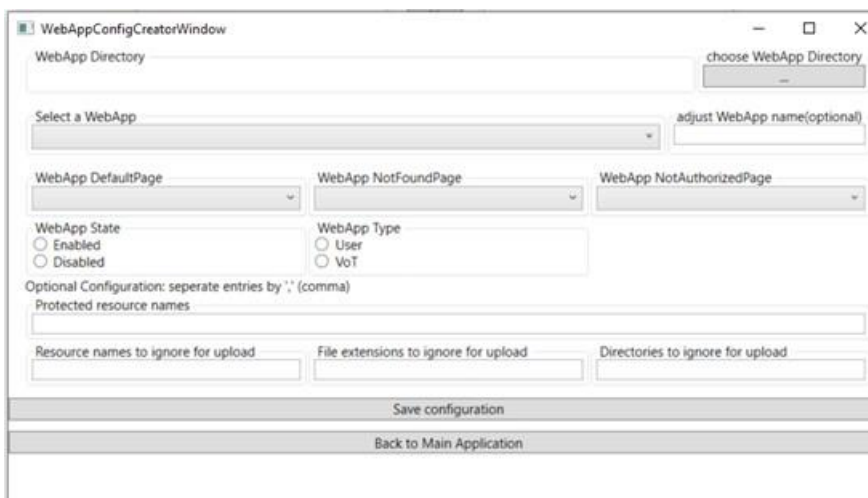


Figure 3-7: Web Application Manager configurator creator window.

2. The minimum configuration that needs to be done here is:
 - a. Choose a WebApp Directory
 - b. Select a WebApp: Select the folder where all the content of the web app is stored. It automatically will maintain the subfolder structure.
 - c. Select a DefaultPage.
 - d. WebApp State: Enabled (if you want users to access it).
 - e. WebApp Type: User.
 - f. Save the configuration and go back to the main application

For this application example:

Select a WebApp: PLCLoad

adjust WebApp name(optional):

WebApp DefaultPage: public/startPage.html

WebApp NotFoundPage:

WebApp NotAuthorizedPage:

WebApp State: ☒ Enabled ☐ Disabled

WebApp Type: ☒ User ☐ VoT

Optional Configuration: separte entries by ',' (comma)

Protected resource names:

Resource names to ignore for upload:

File extensions to ignore for upload:

Directories to ignore for upload:

Save configuration

Back to Main Application

Figure 3-8: Web Application Manager configuration for this application

3. Edit Plc Rack configuration(s) or create config: Create a configuration for the PLC or rack of PLCs where you want to download the application. Proceed as follows:
 - a. First add a name to the Plc rack and then click on Add button.

PlcRackConfigCreatorWindow

Plc Rack Name: plc1

Add Rack Configuration

Add

Select a Rack to configure:

Rack Plc IPs/Dns Names:

Save configuration

Back to Main Application

Figure 3-9: Web Application Manager PLC rack configuration creator window

- b. The next step will be to select the rack from the list of racks and give it the list of IPs that you want to address separated by commas. For example:

PlcRackConfigCreatorWindow

Plc Rack Name:

Add Rack Configuration

Add

Select a Rack to configure: plc1

Rack Plc IPs/Dns Names: 192.168.0.1, 192.168.0.10

Save configuration

Back to Main Application

Figure 3-10: Web Application Manager PLC rack configuration for this application

- c. Save the configuration and go back to the main application.

4. Now is time to perform the download of the application(s) to the PLC(s):

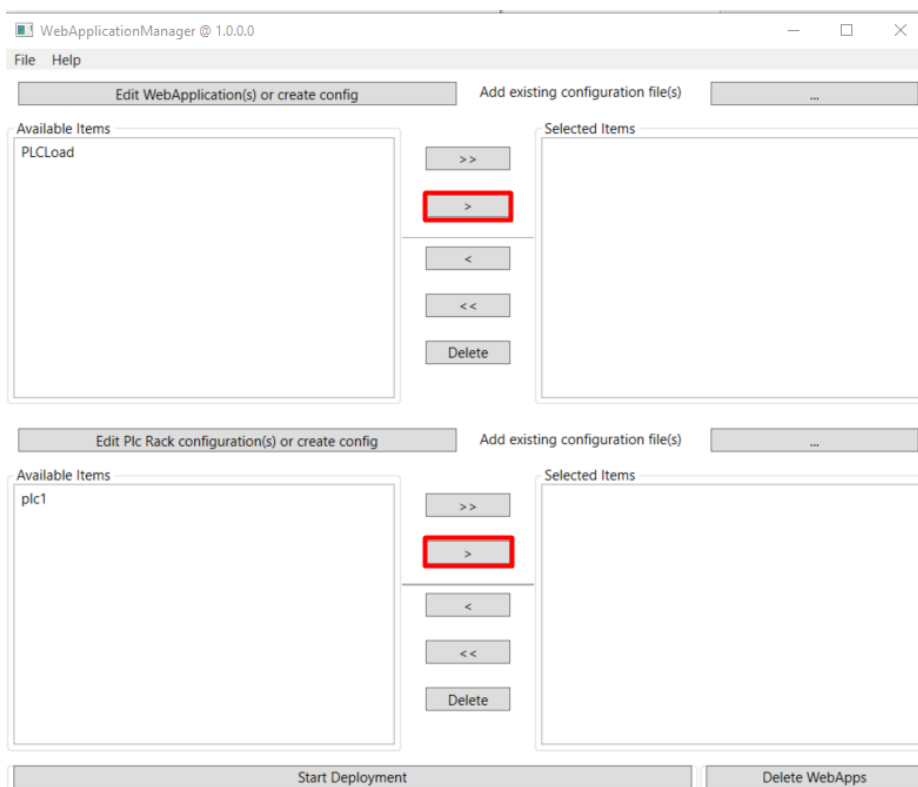


Figure 3-11: Web Application Manager Download Process 1

What is going to be transferred and the place that will be directed needs to be on the right side of the main application. To do that, select the objects on the available items list, click on the ">" symbol. The result will be:

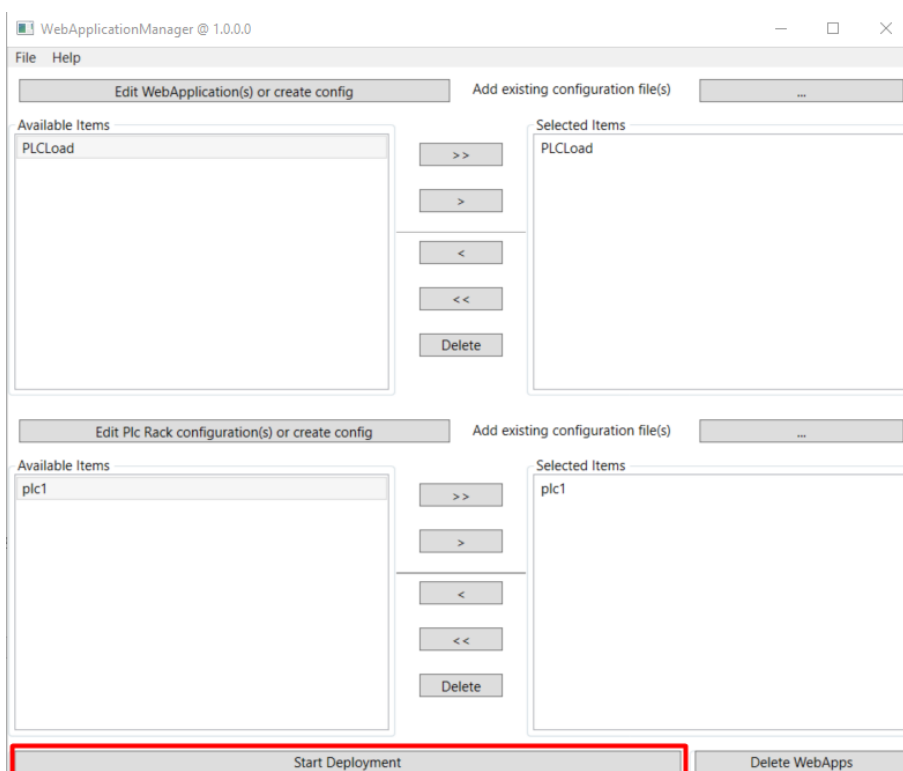


Figure 3-12: Web Application Manager Download Process 2

Now you can click on Start Deployment. You will be redirected to a form that will request a user and password with enough rights (manage user defined web pages is mandatory):

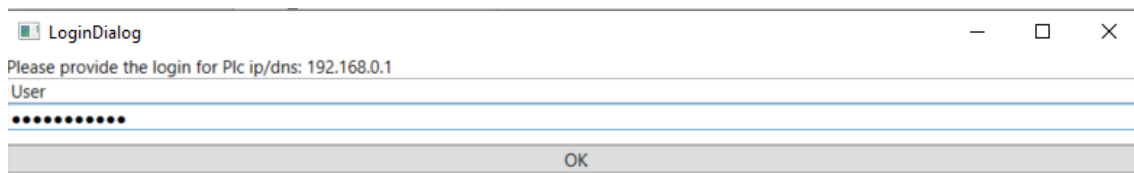


Figure 3-13: Web Application Manager Login dialog

Clicking on OK will start the download process automatically.

NOTE

Note that if an application has been previously downloaded to a PLC it is possible also to delete it from here following the same procedure for downloading but clicking on the "Delete WebApps" button.

NOTE

It is possible to select multiple PLC racks and multiple web applications to perform a batch download to different rack configurations with a single click.

Access via HTTPS

With the URL <https://ww.xx.yy.zz>, you get access to the standard web pages, where ww.xx.yy.zz corresponds to the IP address of the interface of the SIMATIC S7-1500/ S7-1200 G2 CPU.

NOTE

For the S7-1200 G2 web server it is only possible to access through the Web API interface, so there is no option to enable or disable the "Permit access only with HTTPS" or "Permit data access only via Web API" option.

HTTPS is used to encrypt all communication between the web browser and web server. If the check box "Only allow access via HTTPS" is checked, the web pages can only be accessed via HTTPS.

Log in

In the user list available under the Security Settings > User and Roles you will find two pre-configured users: UserAdministrator and User.

The "UserAdministrator" user has the Engineering Administrator role, and it is used for opening the project (the project has been previously configured as protected). The password for accessing the project is "Siemens123!".

The "User" is configured with a user defined Role called "role" that has the following runtime rights assigned for each PLC on the project:

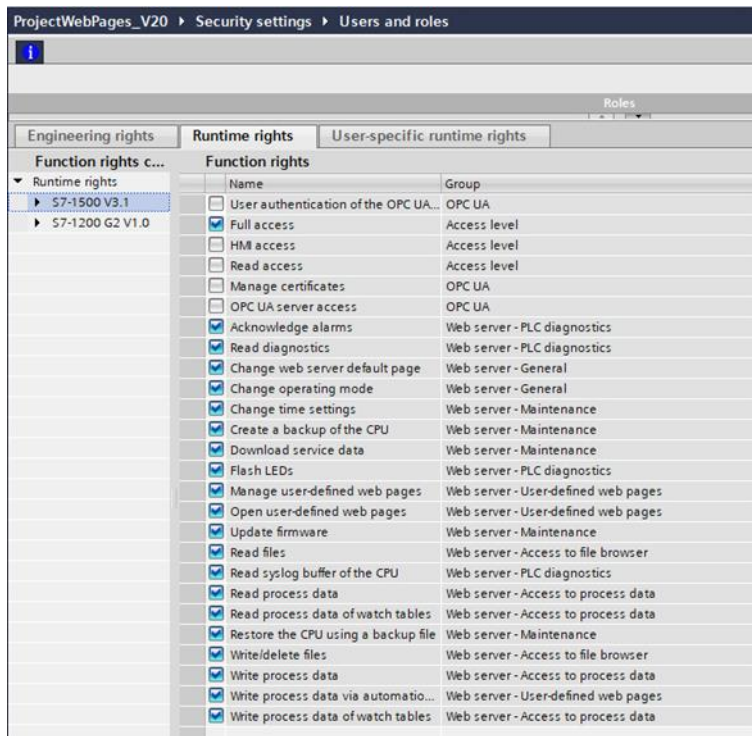


Figure 3-14: User runtime rights for S7-1500

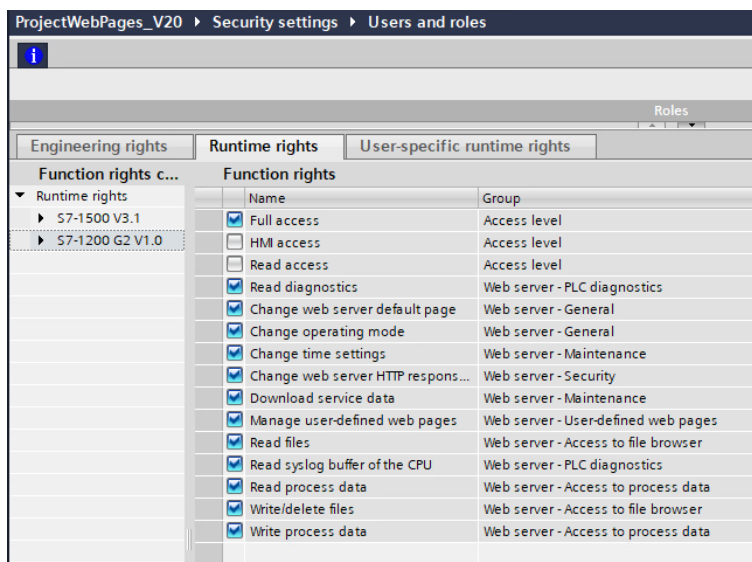


Figure 3-15: User runtime rights for S7-1200 G2

Users and Passwords

Application	User	Password
TIA Portal – Open Project	UserAdministrator	Siemens123!
TIA Portal – Download Project	User	Siemens123!
User Defined Web Page	User	Siemens123!

Table 3-4

4. Operating the Application Example

For a S7-1500 there are two options, the first option is not available in S7-1200 G2.

The web server of SIMATIC S7-1500 / S7-1200 G2 already offers a lot of information about the respective CPU via integrated standard web pages.

You can find a detailed description of the setup of the standard web pages in the manual:

<https://support.industry.siemens.com/cs/ww/en/view/109977246>

Open the user-defined page as follows:

1. Start a web browser, such as Google Chrome.

For the address enter, the IP address of the CPU, plus a "~", plus the name of the application, in this case: "myApp", e.g. <https://192.168.0.1/~myApp>

NOTE

Right now, for the FW V1.0 version of the S7-1200 G2, there are no standard web pages available on this family of devices to access through the system web pages and the user-defined web pages option of the menu.

2. To start the example, click "Homepage of the application myApp".

The "Login" web page opens.

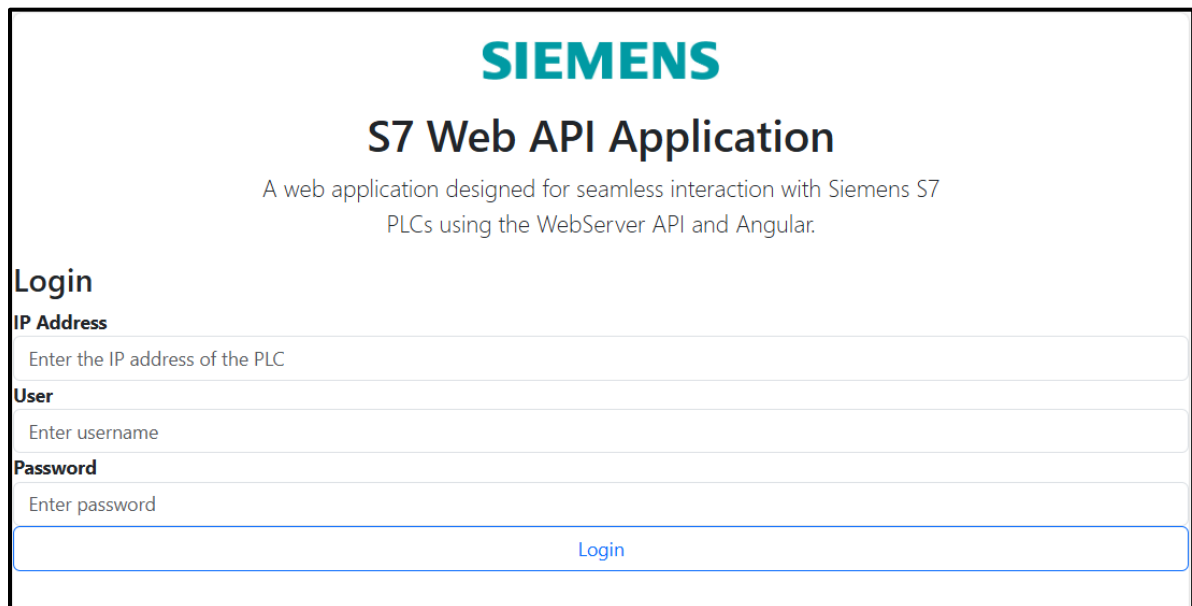


Figure 4-1: Web application login page

Now the user is requested to perform a Web API session by logging in with the username and password. In this case the user used is the same as the previous one, with the following login data:

- IP address: The IP address of the interface you set (e.g. 192.168.0.1)
- Username: User
- Password: Siemens123!

Click the Login button. A popup message will appear confirming that you were successfully logged in.



Figure 4-2: Web application login confirmation

3. Now the "Landing Page" web page opens, that by default displays the "Overview Tank" web page.
4. On the left side of the page, you can choose between the different web pages that are part of the application.



Figure 4-3: Web application navigation bar

4.1. Operating the web page "Tank Overview"

The "Overview Tank" component in the application is an interactive interface where operators can control and monitor tank operations using a visual and functional UI. Here is the detailed workflow of what happens with each control (i.e., button) interaction:

Visual Elements Displayed:

- **Tank Image:** This dynamic image reflects the current state of the tank (either filling, emptying, or idle).
- **Data Values Displayed:**
 - **Flowrate:** Current flow rate of the liquid into or out of the tank.
 - **Tank Level:** Current level of liquid in the tank.
 - **Measurement Units:**
 - **Tank Level Overflow:** Maximum sensor level for overflow.
 - **Tank Level Maximum:** Maximum reasonable tank level.
 - **Tank Level Mid:** Mid-point level of the tank.
 - **Tank Level Minimum:** Minimum reasonable limit for tank level before the lack warning.
 - **Tank Level Lack:** Sensor level indicating insufficient volume.

Control Buttons

1. Start Button:

- a. Functionality: Begins the process of filling the tank.
- b. Image Update: Changes the tank image to show the tank is filling.
- c. Tank Level Value: Begins to increase as the tank fills.
- d. Further Action: While in this state, the valve is closed by default. You can open the valve using the "Open Valve" button to switch to emptying the tank.

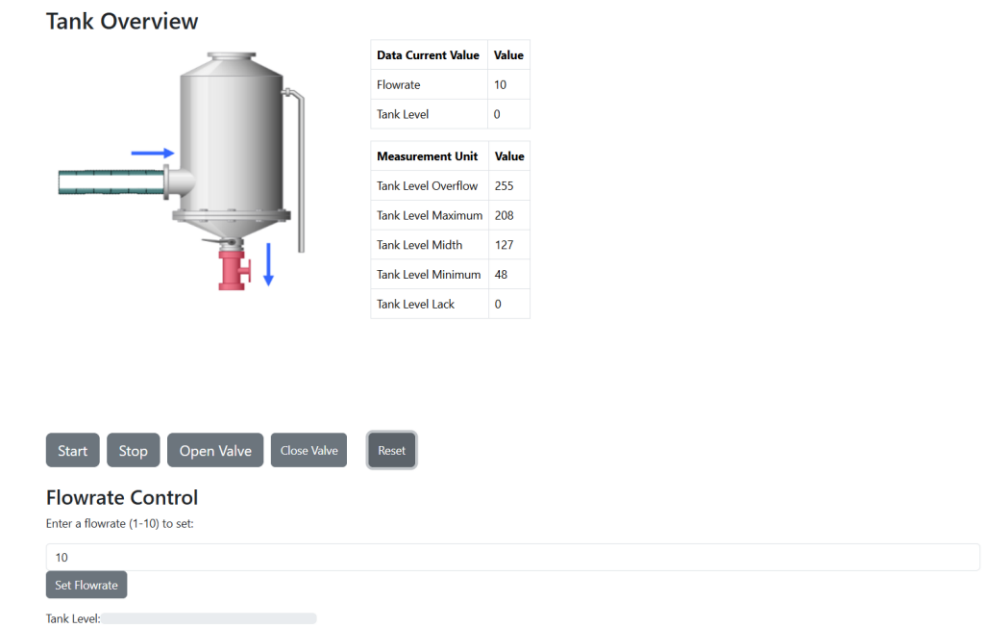


Figure 4-4: Web application Overview Tank. Stop State

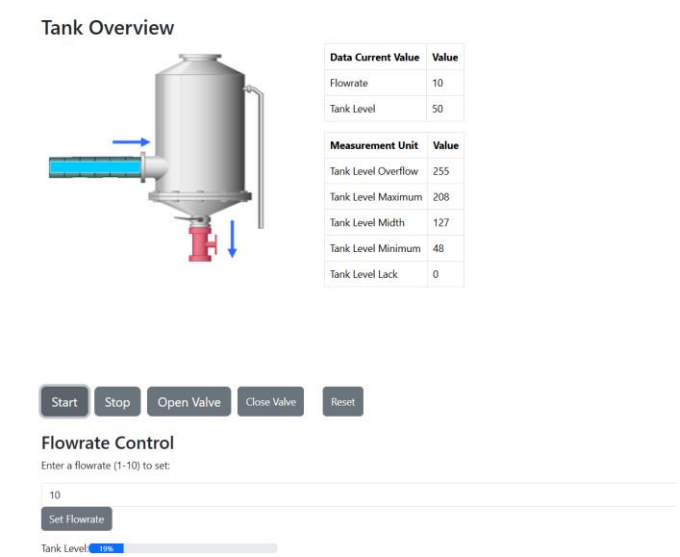


Figure 4-5: Web application Overview Tank. Filling State

2. Open Valve Button:

- a. Functionality: Opens the tank valve to start emptying the tank.
- b. Image Update: Updates the image to reflect that the tank is emptying. See in [Figure 4-6](#)
- c. Tank Level Value: Decreases as the liquid is released.
- d. Toggle Action: If the "Close Valve" button is pressed, the tank returns to the filling state, updating the image and reversing the tank level change rate.

3. Close Valve Button:

- a. Functionality: Closes the valve, reversing the action from emptying to filling.
- b. Image Update: Changes the tank image back to filling. See in [Figure 4-5](#)
- c. Tank Level Value: Begins to increase again as the tank starts filling.

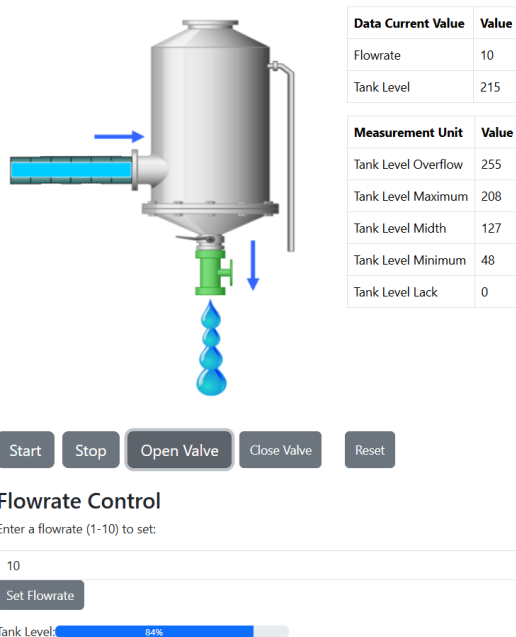
Tank Overview

Figure 4-6: Web application Overview Tank. Emptying State

4. Stop Button:

- a. Functionality: Halts all tank functions; no more filling or emptying occurs.
- b. State Maintenance: Transitions to a steady state where no modifications to tank levels occur; it freezes the current state.
- c. Image Update: Updates the tank image to idle or stopped state indicating no active changes. See in [Figure 4-4](#)

5. Reset Button:


- a. Functionality: Resets the plant operations, specifically resetting the tank level to zero.
- b. Automatic actions: After hitting reset, the system also defaults back to the "Stop" state, as resetting typically precedes a stop in operation to ensure safe reinitialization.

Flow Rate Setting Form

- **Input for Flow rate:** Allows users to enter a desired flow rate between 1 and 10, which might control the speed at which the tank fills or empties.
- **Set flow rate button:** Upon clicking this button after entering a value, the application updates the actual flow rate being used in the plant operations.

Dynamic Display of Tank Filling Level

- **Filling level progress bar:** visually representing the tank's fill level in percentage, this dynamic bar adjusts automatically, expanding or contracting based on the current state of the plant.

A web form titled "Flowrate of Tank Filling/Emptying". It contains a text input field with the value "4" and a blue button labeled "Set Flowrate". Below the button, it displays "Actual Flowrate: 4".

Flowrate of Tank Filling/Emptying

Enter a flowrate (1-10) to set:

4

Set Flowrate

Actual Flowrate: 4

Figure 4-7: Web application Plant Status. Flowrate form

4.2. Operating the web page "Data View"

The "DataView" component of the application serves as a real-time data monitoring interface, specifically designed to track and display varying measurements such as performance metrics, system states, or environmental conditions over time.

Interface and Display Elements:

- **Data Table:** The primary element of the DataView is the data table that displays two key pieces of information for each entry:
 - **Timestamp:** The exact time at which the data was recorded.
 - **Value:** The recorded measurement or data point corresponding to that timestamp.

Data Capture and Update Frequency:

- **Initial Data Capture:** When the DataView component is first loaded or activated, it immediately captures the initial set of data points from the associated sensors or system monitors, displaying them with the precise timestamps.
- **Subsequent Updates:** The component is configured to update its data at specific intervals to reflect new measurements:
 - **Every 10 seconds:** The primary interval at which new data is captured and displayed.
 - **Every 2 seconds:** A more frequent update interval that may be triggered under certain conditions or during specific operational phases where closer monitoring is required.

Data View		
<i>Data updated every 2 seconds</i>		
#	Timestamp	Value
1	2025-01-14/11:32:19.9893	4
2	2025-01-14/11:32:20.9909	8
3	2025-01-14/11:32:21.9918	12
4	2025-01-14/11:32:22.9925	16
5	2025-01-14/11:32:23.9930	20
6	2025-01-14/11:32:24.9945	24
7	2025-01-14/11:32:25.9951	28
8	2025-01-14/11:32:26.9966	32
9	2025-01-14/11:32:27.9971	36
10	2025-01-14/11:32:28.9986	40
11	2025-01-14/11:32:29.9990	44
12	2025-01-14/11:32:31.0005	48
13	2025-01-14/11:32:32.0013	52
14	2025-01-14/11:32:33.0020	56
15	2025-01-14/11:32:34.0036	60
16	2025-01-14/11:32:35.0043	64
17	2025-01-14/11:32:36.0058	68
18	2025-01-14/11:32:37.0065	72
19	2025-01-14/11:32:38.0079	76

Figure 4-8: Web application Data View

4.3. Operating the web page "Diagnostic Buffer"

The "Diagnostic Buffer" component in the system functions as a comprehensive diagnostic log viewer, dedicated to displaying detailed records of system events, warnings, and errors. This component is crucial for system maintenance, troubleshooting, and ensuring operational transparency.

Core Display Elements

- **Last Modified Timestamp:** Displays the timestamp of the latest entry added to the diagnostic buffer.
- **Total Count of Entries:** Indicates the number of diagnostic entries that are currently available in the buffer.
- **Maximum Capacity:** Shows the total storage capacity of the diagnostic buffer.

Diagnostic table

This table is the central feature of the component and includes the following columns:

- **Timestamp:** The exact time the event was logged.
- **Status:** An indicator or status code representing the nature (severity, type) of the event.
- **Long Text:** A detailed description of the event.
- **Short Text:** A concise summary of the event.
- **Help Text:** Additional help or guidance related to the event.
- **Event Text List ID:** Identifies the text list this event belongs to.
- **Event Text ID:** Identifiers the text ID of the text list.

Control Features

- **Refresh Button:** Allows users to manually update the table to display the latest logs from the diagnostic buffer. Clicking this button triggers a new fetch request to the PLC to retrieve the most up-to-date records.

PLC Diagnostic Buffer						
Last Modified: 2025-01-14T10:49:31.370090852Z						
Current Count: 22 / Max Count: 3200						
Refresh						
Timestamp	Status	Long Text	Short Text	Help Text	Event Text List ID	Event Text ID
1/14/2025, 11:49:31 AM	incoming	CPU info: Follow-on operating mode change Power-on mode set: WARM RESTART to RUN (if CPU was in RUN before power off) Pending startup inhibit(s): - No startup inhibit set CPU changes from STARTUP to RUN mode PLC_1 / PLC_1	Follow-on operating mode change - CPU changes from STARTUP to RUN mode	Implicit operating state transition request from operating system.Note the additional information in the diagnostics buffer entry, particularly the reaction (mode transition actually initiated and newly set startup inhibit conditions).	2	16396
1/14/2025, 11:49:31 AM	incoming	CPU info: Follow-on operating mode change Power-on mode set: WARM RESTART to RUN (if CPU was in RUN before power off) Pending startup inhibit(s): - No startup inhibit set CPU changes from STOP to STARTUP mode PLC_1 / PLC_1	Follow-on operating mode change - CPU changes from STOP to STARTUP mode	Implicit operating state transition request from operating system.Note the additional information in the diagnostics buffer entry, particularly the reaction (mode transition actually initiated and newly set startup inhibit conditions).	2	16396
1/14/2025, 11:49:31 AM	incoming	CPU info: Follow-on operating mode change Power-on mode set: WARM RESTART to RUN (if CPU was in RUN before power off) Pending startup inhibit(s): - No startup inhibit set CPU changes from STOP (initialization) to STOP mode PLC_1 / PLC_1	Follow-on operating mode change - CPU changes from STOP (initialization) to STOP mode	Implicit operating state transition request from operating system.Note the additional information in the diagnostics buffer entry, particularly the reaction (mode transition actually initiated and newly set startup inhibit conditions).	2	16396
1/14/2025, 11:49:31 AM	incoming	CPU info: Technology package MC Base loaded: Version: V8.0.0 PLC_1 / PLC_1	Technology package MC Base loaded: Version: V8.0.0 -		2	16778
1/14/2025, 11:49:31 AM	incoming	CPU info: Boot up memory card type: Program card (external load memory) CPU changes from OFF to STOP (initialization) mode PLC_1 / PLC_1	Boot up - CPU changes from OFF to STOP (initialization) mode	The CPU is booting up. Note the additional information in the diagnostics buffer entry, particularly the reaction (mode transition actually initiated and newly set startup inhibit conditions).	2	16385
1/14/2025, 11:47:59 AM	incoming	CPU info: Shut down CPU changes from RUN to OFF mode PLC_1 / PLC_1	Shut down - CPU changes from RUN to OFF mode	The CPU is shutting down.Note the additional information in the diagnostics buffer entry, particularly the reaction (mode transition actually initiated and newly set startup inhibit conditions).	2	16386

Figure 4-9: Web application Diagnostic Buffer

4.4. Operating the web page "Syslog Buffer"

The "Syslog Buffer" component in the application is designed to provide insights into the system's operational logs. This component is for understanding the system's activities, including any anomalies, informational messages, or errors that systems typically log for operational transparency and troubleshooting purposes.

Core Display Elements

- **Total Count of Entries:** Shows the number of syslog entries currently stored.
- **Lost Count:** Indicates the number of log entries that were lost or could not be stored, due to buffer overflows or transmission errors.

Syslog Table

- **Raw Entry:** Each row in the table shows a raw log entry exactly as it is stored in the system's log buffer.

PLC Syslog Buffer	
Total Count: 21	
Lost Count: 0	
Raw Entry	
{"raw":{"raw":"<29> 1 2025-01-14T11:34:01.488Z 192.168.0.1 StateManager - ID98 [device@4329.6.100.1.1500 devVendor=\"Siemens\" devProduct=\"CPU1500 (PLCSIM)\" FWVersion=\"S31.18.65\"][function@4329.6.100.1.1500 fct=\"change operating mode\" oldState=\"RUN\" newState=\"STOP\"] SE_OPMOD_CHANGED"}}	
{"raw":{"raw":"<29> 1 2025-01-14T10:52:01.296Z 192.168.0.1 Webserver - ID307 [device@4329.6.100.1.1500 devVendor=\"Siemens\" devProduct=\"CPU1500 (PLCSIM)\" FWVersion=\"S31.18.65\"][session@4329.6.100.1.1500 protocolType=\"HTTPS\" userName=\"User\" src=\"192.168.0.4\" sessionId=\"4\"][function@4329.6.100.1.1500 fct=\"logout\" errReason=\"session timeout\"] SE_SESSION_TERMINATED"}}	
{"raw":{"raw":"<29> 1 2025-01-14T10:52:01.296Z 192.168.0.1 Webserver - ID307 [device@4329.6.100.1.1500 devVendor=\"Siemens\" devProduct=\"CPU1500 (PLCSIM)\" FWVersion=\"S31.18.65\"][session@4329.6.100.1.1500 protocolType=\"HTTPS\" userName=\"User\" src=\"192.168.0.4\" sessionId=\"3\"][function@4329.6.100.1.1500 fct=\"logout\" errReason=\"session timeout\"] SE_SESSION_TERMINATED"}}	
{"raw":{"raw":"<29> 1 2025-01-14T10:52:01.296Z 192.168.0.1 Webserver - ID307 [device@4329.6.100.1.1500 devVendor=\"Siemens\" devProduct=\"CPU1500 (PLCSIM)\" FWVersion=\"S31.18.65\"][session@4329.6.100.1.1500 protocolType=\"HTTPS\" userName=\"User\" src=\"192.168.0.4\" sessionId=\"6\"][function@4329.6.100.1.1500 fct=\"logout\" errReason=\"session timeout\"] SE_SESSION_TERMINATED"}}	
{"raw":{"raw":"<29> 1 2025-01-14T10:52:01.296Z 192.168.0.1 Webserver - ID307 [device@4329.6.100.1.1500 devVendor=\"Siemens\" devProduct=\"CPU1500 (PLCSIM)\" FWVersion=\"S31.18.65\"][session@4329.6.100.1.1500 protocolType=\"HTTPS\" userName=\"User\" src=\"192.168.0.4\" sessionId=\"1\"][function@4329.6.100.1.1500 fct=\"logout\" errReason=\"session timeout\"] SE_SESSION_TERMINATED"}}	
{"raw":{"raw":"<29> 1 2025-01-14T10:52:01.296Z 192.168.0.1 Webserver - ID307 [device@4329.6.100.1.1500 devVendor=\"Siemens\" devProduct=\"CPU1500 (PLCSIM)\" FWVersion=\"S31.18.65\"][session@4329.6.100.1.1500 protocolType=\"HTTPS\" userName=\"User\" src=\"192.168.0.4\" sessionId=\"8\"][function@4329.6.100.1.1500 fct=\"logout\" errReason=\"session timeout\"] SE_SESSION_TERMINATED"}}	

Figure 4-10: Web application Syslog Buffer.

4.5. Operating the web page "Alarms"

The "Alarms" component within the application serves as a central hub for monitoring and managing alarms generated by the system. It is crucial for operational safety and efficiency, providing detailed insights into system alerts that may indicate malfunctions, warnings, or the need for maintenance.

Display Elements

- **Last Modified Timestamp:** Shows when the most recent alarm was modified or added, providing a time reference for the latest alarm activities.
- **Current Alarms Count:** Displays the number of active alarms currently being reported by the system.
- **Maximum Alarms Count:** Indicates the capacity limit of how many alarms can be simultaneously active.

Alarm Table

This central table presents a detailed list of alarms, including:

- **ID:** Unique identifier for each alarm.
- **Timestamp:** The exact time the alarm was raised.
- **Alarm Text:** Descriptive text explaining the alarm.
- **Status:** The status of the alarm (e.g., active, acknowledged, resolved).
- **Acknowledgment:** Indicates whether the alarm has been acknowledged by a user, which means it has been seen and is either being addressed or has been deemed non-critical.

Control Features

- Alarm Acknowledgment Form:
 - **Functionality:** Allows users to enter the ID of an alarm they wish to acknowledge.
 - **Acknowledge Alarm Button:** Upon entering a valid alarm ID and clicking this button, the specified alarm is marked as acknowledged in the system, changing its status in the alarm table.

Alarms View

Last Modified:

Current Alarms: 0

Max Alarms: 0

ID	Timestamp	Alarm Text	Status	Acknowledgement
Alarm ID				
<div>Enter Alarm ID</div>				
<div>Acknowledge Alarm</div>				

Figure 4-11: Web application Alarms.

5. Function Mechanisms of this Application Example

Using a tank system as an example, it is shown how to exchange data between the web server of the CPU and a computer with just a web browser. The entry point of the web application is the "index.html" file located in the src folder of the project myApp.

Automatically, when you launch the built project, you will be redirected to the login form. After a successful login, you will be redirected by default to the "LandingPage" page and you will have a navigation bar on the left side that contains this list of web pages:

- Overview Tank
- Data View
- Plant Status
- Diagnostic Buffer
- Syslog Buffer
- Alarms

The frontend framework used to manage this web pages is Angular, where a single page can load multiple web pages without the need of reloading the web browser. Each html page has a single typescript script with the same name for an easier understanding and to modularize the programming to be able to export parts of this project to a different project.

The interaction of this web application with the Web API of the PLC has been developed using a siemens opensource package that has been published on npmjs: [@siemens/simatic-s7-webserver-api - npm](https://www.npmjs.com/package/@siemens/simatic-s7-webserver-api)

5.1. Functional Principle of the S7 User Program

The S7 user program of this application example only serves to illustrate individual functionalities of SIMATIC STEP 7 (TIA Portal).

The following Figure shows the call structure in the S7 user program.

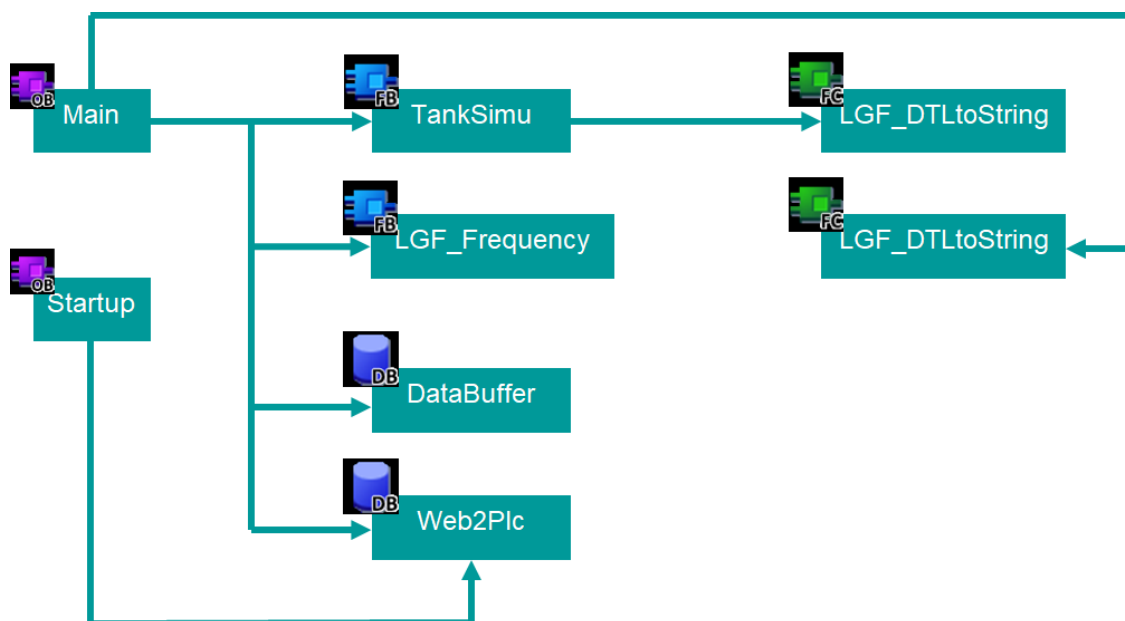


Figure 5-1: PLC program structure

The following tags are used in the data blocks "Web2Plc" and "DataBuffer":

web2plc		
	Name	Data type
1	Static	
2	tankLevelLack	Int
3	tankLevelMinimum	Int
4	tankLevelMidth	Int
5	tankLevelMaximum	Int
6	tankLevelOverflow	Int
7	tankLevel	Int
8	tankLevelScale	Int
9	flowrate	Int
10	statusValveCPU	Bool
11	stop	Bool
12	start	Bool
13	reset	Bool
14	startStop	Bool
15	alarmText	Int
16	alarmColor	Int
17	actFlowrate	Int
18	closeValve	Bool
19	openValve	Bool

Figure 5-2: DB "Web2Plc"

DataBuffer		
	Name	Data type
1	Static	
2	data	Array[0..20] of "typeDataStruct"

Figure 5-3: "DataBuffer"

typeDataStruct		
	Name	Data type
1	value	Int
2	timeStamp	String[24]

Figure 5-4: PLC data type "typeDataStruct"

5.1.1. Startup (OB100)

In OB "Startup" (OB100), a start value for the flow rate "web2plc.flowrate" and the limit values of the tank shape tags are stored: "web2plc.tankLevel", "web2plc.tankLevelLack", "web2plc.tankLevelMinimum", "web2plc.tankLevelMidth", "web2plc.tankLevelMaximum" and "web2plc.tankLevelOverflow".

5.1.2. Main (OB1)

In OB "Main" (OB1) there is a set of functions:

1. Copy the input value flow rate to the actual flow rate.
2. Set the "StartStop" tag with the inputs "Start", "Stop" and "Reset".
3. Force the valve to be closed when the system is in stop state.
4. Set the "StatusValveCPU" tag with the inputs "OpenValve", "CloseValve" and "Reset".
5. When the input reset is enabled reset the tank level and the inputs "Start" and "Stop".
6. Call and set the trigger FB to have a cyclical input enabled every second.
7. Call the tank simulation once per second.

5.1.3. TankSimu (FB1)

How the FB "TankSimu" works

In the FB "TankSimu", the filling or emptying of a tank is simulated, depending on the flow rate and the valve position.

The block is run through once per second.

On the web page, you determine the flow rate via the tag "web2plc.flowrate". The tank level is increased or decreased with the flow rate when the FB "TankSimu" is called up. The current fill level is stored in the PLC tag "tankLevel".

The valve position is read and stored in the CPU in the PLC tag "web2plc.statusValveCPU" via the two PLC tags "web2plc.openValve" and "web2plc.closeValve".

StartStop Status

The tank level is only changed, and values are only entered into the ring buffer if the "web2plc.startStop" bit is set.

Valve Status

Via the "web2plc.statusValveCPU" bit, the most recently pressed button (OpenValve or CloseValve) is saved.

Depending on this bit, the tank is either emptied or filled.

Fill Tank

The filling of the tank starts with a query to check if the tank is already full. If the tank is not full, the tank level increases with the flow rate. The tank level is limited by the "web2plc.tankLevelOverflow" value.

Empty Tank

The emptying of the tank is like the filling of the tank. The tank level is reduced with the flow rate and is limited to 0.

5.2. Functionality of the HTML Files with Typescript

The following chapter explains the interaction between HTML files and their respective TypeScript counterparts within the Angular framework, specifically focusing on the project's structure shown under the "src" folder. The structure primarily comprises components, each responsible for rendering a specific part of the application's user interface and handling user interactions.

In this example, to correctly display a web page, e.g., the Data View, following the Angular architecture, the component requires: a .ts file (TypeScript), a .html file (template), and a .css file for styles. These files are typically grouped within a subfolder under the /app/components/ directory. For example, the Data View functionality is managed through the dataView.component.ts and dataView.component.html files located in the /dataView folder.

5.2.1. Typescript-SIMATIC-s7-webserver-api

This package is an opensource package developed by Siemens and published in npm and github repositories. It is used to manage the requests and responses of the web application to the web server of the PLC.



Reference to repository: [GitHub - siemens/typescript-simatic-s7-webserver-api](https://github.com/siemens/typescript-simatic-s7-webserver-api)

There is publicly available documentation about how to use this package with an example step by step explanation.

In this project, in every typescript file there are multiple calls to functions available to this package that will be explained in each section that applies.

The common features that are needed in all communications is the configuration object that is compounded of:

```
// Request configuration
let config = new RequestConfig();
config.protocol = "https";
config.verifyTls = false;
config.address = sessionStorage.getItem('plcAddress') || '';
```

Where RequestConfig is the class defined on the library that has these three mandatory components (it is possible to include here a certificate to verify the identity of the client that is sending the request).

Thanks to angular it is possible to store the "plcAddress" included in the form of the startPage.html with the getItem() method.

5.2.2. Angular

Angular, often referred to as Angular 2+, is an open-source web application framework led by the Angular Team at Google and a community of individuals and corporations. Angular is a complete rewrite from the same team that built AngularJS. It is primarily aimed at building dynamic client-side applications in HTML and TypeScript (though it can also handle JavaScript).

Usage and Core Features

1. **Component-Based Architecture:** Angular introduces a hierarchical component-based framework that ensures applications are modular. This modularity allows for reusable components, easier maintenance, and improved scalability. Components encapsulate the data binding, logic, and template rendering into reusable blocks.
2. **TypeScript-Based:** Angular applications are built using TypeScript, a superset for JavaScript, which ensures higher security as it supports static typing along with great tooling and IDE support. Static typing makes it easier to maintain larger code bases and prevent potential runtime errors.
3. **Reactive Programming Support:** Angular leverages RxJS (Reactive Extensions for JavaScript), enabling efficient and sophisticated composition of asynchronous operations and event-based programs. This is pivotal for managing a vast number of streaming data and can be fused seamlessly with Angular's Event Model.
4. **Two-Way Data Binding:** Two-way data binding continues to be a strong feature in Angular, ensuring that the model state reflects the inputs in the application UI and vice-versa without necessitating additional effort for syncing data.
5. **Dependency Injection (DI):** Dependency injection in Angular allows developers to define dependencies of classes simply and declaratively. This technique significantly aids in creating decoupled, maintainable, and testable code. DI is particularly useful for managing services where a class requests dependencies from external sources rather than creating them internally.
6. **Ahead-of-Time (AoT) Compilation:** Angular supports AoT compilation, which compiles the application components and templates to native JavaScript and HTML before the browser downloads and runs the code. This practice improves the rendering speed and security of the application.
7. **Angular CLI:** The Angular Command Line Interface (CLI) is a robust toolset that automates and assists in developing Angular applications. It manages tasks such as initialization, configurations, component scaffolding, development, and deployment.
8. **Router Module:** The Angular Router enables navigation between different components that make up an application. It works by interpreting a browser URL as an instruction to navigate to a client-generated view. It is crucial for mimicking the navigation of a multi-page application without the need to leave the single-page application environment.

Advantages of Angular

- **Productivity and Tooling:** Offers excellent tools and IDEs. The Angular CLI streamlines application generation, component scaffolding, configuration, and code deployment.
- **Performance:** Implements optimized change detection and powerful rendering capabilities.
- **Universal:** Supports server-side rendering for Angular apps.
- **Mobile & Desktop:** Can build applications across web, mobile web, native mobile, and native desktop.
- **Full Development Story:** Supports testing, animation, accessibility, and other features natively.

Angular has streamlined and refined the concept of a single-page application beyond its original inception in AngularJS. Its architecture is designed to support robust, large-scale applications and encourages an environment conducive to maintenance and scalability using a full suite of development tools.

5.2.3. Style sheet: Bootstrap

Bootstrap is a popular open-source front-end framework used to design and build responsive, mobile-first web applications. It provides pre-designed components such as buttons, forms, modals, and navigation bars, along with a powerful grid system to create layouts that adapt to different screen sizes. Bootstrap helps developers save time by offering consistent styles and functionality, making it easier to create visually appealing and functional websites without starting from scratch.

Bootstrap comes packed with utility classes that speed up the development process by reducing the need for writing custom CSS. These utilities cover a wide range of common design tasks such as margin and padding adjustments, text alignment, color, background modifications, and more. These are designed to be as modular as possible, allowing developers to quickly assemble pages with fine control over layout and design.

While Bootstrap provides an extensive set of pre-designed components, it also supports the creation of custom components. This can be particularly beneficial when the default components need to be modified or extended to fit specific requirements better.

bootstrap.min.js

The file `bootstrap.min.js` is the minified version of the Bootstrap JavaScript library. It includes the JavaScript code that powers interactive components like dropdown menus, modals, tooltips, and carousels. The minification process reduces the file size by removing unnecessary characters like spaces and comments, improving page load times for production environments.

bootstrap.min.js vs bootstrap.min.css

On the other hand, `bootstrap.min.css` is the minified version of the Bootstrap CSS file. It contains the styles and layout definitions for the framework, controlling the appearance of elements such as typography, colors, spacing, and responsiveness. Like the JavaScript file, it is optimized for production use with a reduced file size.

The key difference is that `bootstrap.min.js` handles the interactive and dynamic behavior of components, while `bootstrap.min.css` manages their visual presentation and layout. Both are typically used together to fully implement Bootstrap's functionality in a web project.

5.2.4. Webpack in Angular projects

Webpack is a powerful module bundler that plays a pivotal role in modern web development, particularly when preparing applications for deployment. It orchestrates the transformation and bundling of various assets like JavaScript, TypeScript, CSS, and images into optimized files tailored for browser compatibility.

Transpilation and Module Handling

One of Webpack's key functionalities is transpiling TypeScript into JavaScript, enabling the resultant code to run in browsers that do not support TypeScript natively. This process is crucial as it ensures that innovative features implemented in TypeScript are accessible across all browsers without compatibility issues.

Webpack also addresses the challenge of handling different module formats that are not natively supported by browsers. It converts modules (e.g., CommonJS, Node.js-style modules) into a format understandable by browsers, such as ES Modules. This feature is especially significant for integrating advanced JavaScript features and third-party libraries seamlessly into projects without compatibility concerns.

Custom Configuration in Angular

In typical scenarios facilitated by Angular CLI, Webpack is configured implicitly to streamline the development experience. However, in certain sophisticated scenarios such as incorporating specific Siemens libraries (e.g., `@siemens/simatic-s7-webserver-api`) into an Angular project, there arises a need for a customized Webpack configuration.

A custom Webpack configuration is employed to accommodate specific libraries that the default Angular CLI setup does not natively support. The standard configuration provided by Angular wraps Webpack to handle typical applications efficiently but may require extensions to manage more complex dependencies.

Key Webpack Configurations (webpack.config.js)

The webpack.config.js serves as the configuration file for Webpack, defining how the bundler should process and organize the project files. The key sections in the configuration file are:

1. **Entry:** defines the initial starting points for building the dependency graph. For projects involving intricate dependencies or large-scale applications, managing entry points is crucial for optimizing the build process.
2. **Output:** Specifies the output location and names for the bundled files. In complex applications, maintaining organized output settings can significantly aid in deployment and runtime performance.
3. **External:** specifies dependencies that should not be bundled into the output files. Instead, Webpack treats these dependencies as external and assumes that they will be provided by the runtime environment.
4. **Resolve:** defines how webpack should handle file extensions and module imports, for example how to substitute modules that are not compatible with browsers for ones that will work.
5. **Plugins:** extend webpack capabilities by adding specific tools or behaviors to build the process.
6. **Mode:** specifies whether you are in development or production mode, which affects optimizations like minification.

5.3. Web Application Structure

5.3.1. Main Configuration

5.3.1.1. Typescript configuration

Tsconfig.json

This is the base TypeScript configuration file for your project. It specifies how the TypeScript compiler (tsc) should transform TypeScript code into JavaScript.

- **compilerOptions:** This section configures the behavior of the compiler:
 - *target:* Compiles TypeScript into ECMAScript 2022 code, using the latest features supported by modern browsers.
 - *experimentalDecorators:* Enables support for experimental decorators, widely used in Angular for class decoration.
 - *emitDecoratorMetadata:* Provides additional metadata in the output, allowing Angular's dependency injection system to work correctly.
 - *module:* Sets the module system for the project, in this case, to esnext, which supports dynamic imports among other modern JavaScript features.
 - *rootDir:* The root directory of the source files.
 - *moduleResolution:* How TypeScript looks for modules. node is for Node.js-style resolution.
 - *esModuleInterop:* Allows the generation of code to support importing CommonJS modules in compliance with es6 modules.
 - *forceConsistentCasingInFileNames:* Ensures that the casing in module imports matches the case of the filename.
 - *outDir:* Directory to output the compiled JavaScript files.
 - *strict:* Enables a wide range of type checking behavior that results in more robust code.
 - *skipLibCheck:* Skips type checking of default library declaration files.
- **angularCompilerOptions:** Specific to Angular projects, providing additional configurations for the Angular template compiler:
 - *enableIvy:* Enables the next generation rendering engine for Angular.
 - *compilationMode:* The Angular compilation mode (full for complete compilation).
 - *jitMode* and *aot:* Configuration for Just-In-Time (JIT) or Ahead-Of-Time (AOT) compilation.
- **include and exclude:** Define which directories or files should be included or excluded from compilation, focusing on src and excluding node_modules by default.

Tsconfig.app.json

This configuration is tailored for the app's actual runtime Typescript compilation.

- **extends:** Inherits configurations from tsconfig.json.
- **compilerOptions:**
 - *outDir*: Directs the output of the compilation specifically for application files.
 - *baseUrl*: Base directory to resolve non-relative module names.
 - *declaration*: Controls whether to output.d.ts declaration files from TypeScript files. Set to false typically in applications to prevent emitting declarations.
- **files:** Specific entry points for the application, usually main.ts (the main entry point of the Angular app) and polyfills.ts (required polyfills).
- **include:** Ensures type definitions are considered during the compilation.

Tsconfig.spec.json

This configuration is specifically for setting up the Typescript environment for the testing framework (like Jasmine).

- **extends:** Inherits configurations from tsconfig.json.
- **compilerOptions:**
 - *outDir*: Specifies a separate output directory for test compilation.
 - *types*: Declares type definitions to be included, here specifically for Jasmine, enhancing editor support and type safety in tests.

5.3.1.2. Angular main configuration**Angular.json**

The angular.json file is a crucial configuration file in any Angular project. It centralizes the configuration settings for projects built using the Angular CLI. This file helps to define a variety of workspace-wide and project-specific configuration defaults for build, serve, and test tools used by Angular CLI.

Here's a brief breakdown of the key sections within the angular.json of this project:

- **projects:** Defines and configures the applications or libraries in your workspace.
 - *architect*: Specifies "builders" provided or customized for different operations like build, serve, and test. Builders are responsible for understanding how to perform a given task.
 - *serve*: Configuration specific to serving the application, like specifying custom Webpack configuration for development.
 - *build*: Details specifics such as output directory, entry points, and custom Webpack configuration for production builds.
- **cli**: General CLI configuration like analytics permission.
- **schematics**: Presets for the code generation tools used by Angular CLI to turn your commands into operation files.

5.3.1.3. Custom webpack configuration**Custom-webpack.config.js**

This configuration file overrides or extends the default Webpack configuration used by Angular CLI. It provides specific setups, especially when integrating modules that are not natively manageable by Webpack.

The key elements of this file are:

- **plugins**: Additional plugins like ProvidePlugin to polyfill global tags or modules like buffers and processes.
- **resolve**: Configuration to help Webpack locate the fallback module paths effectively, especially for modules not compatible with the browser by default.

5.3.1.4. Build tools configuration

Postcss.config.js

The postcss.config.js file sets up PostCSS plugins for your project. PostCSS is a tool for transforming styles with JS plugins, which can lint your CSS, support tags and mixins, transpile future CSS syntax, inline images, and more.

- **plugins:** This array is where to include any PostCSS plugins you are using. As it is empty here, it means no plugins are currently configured.

5.3.1.5. Linter

ESLint is a static code analysis tool used to identify problematic patterns found in JavaScript code, and it's extensible to TypeScript with appropriate plugins. In Angular projects, setting up ESLint to handle both TypeScript and template files ensures that the entire codebase follows consistent syntax and style rules, making the code cleaner and more maintainable.

The main parts of the eslint.config.js file of this project can be summarized in:

- **ignores:** Specifies paths to ignore during linting, like node_modules and output directories.
- **files:** This config targets all TypeScript files and applies certain plugins and rules.
- **plugins:** Incorporates TypeScript ESLint, import-related linting, and Angular-specific linting.
- **languageOptions** and **parserOptions:** Configurations for how ESLint understands your code, leveraging typescript-eslint and specifying the project file as tsconfig.json.
- **rules:** Defines specific linting rules for TypeScript and Angular. These include enforcing naming conventions, controlling line length, and setting up import order.
- The last block configures ESLint for HTML templates, utilizing Angular-specific template linting rules, enhancing error detection in Angular templates.

This setup not only reinforces good development practices but also aligns code style across multiple developers, crucial for maintaining project quality and readability.

5.3.1.6. Base files

Package.json

The package.json file is a central piece of any Node.js project or any project that uses npm (Node Package Manager) or Yarn for managing packages. It serves multiple essential purposes:

- **Metadata:** Provides basic information about the project like name, version, and description which are important for package management and publication.
- **Manage dependencies:** Lists all the packages that your project depends on to run or develop (divided into dependencies and devDependencies), which allows npm or Yarn to automatically install these packages.
- **Scripting:** Allows the definition of scripts to automate various tasks such as testing, building, and deploying, thereby standardizing workflows and enhancing productivity.
- **Project configurations:** Can include other metadata used by various tools and libraries integrated into the project.

The sections that compound the package.json of this project are:

- **name:** This field specifies the name of the project. It's a vital identifier especially when you want to publish your package to a npm registry.
- **version:** Indicates the current version of the project using semantic versioning. It's crucial for release management and dependency management.

- **scripts:**
 - *ng*: Shortcut to use Angular CLI commands.
 - *start*: Runs the development server for local testing and debugging.
 - *build*: Compiles the application into deployable files usually saved in the *dist/* directory.
 - *watch*: Continuously watches for any changes in the project files and rebuilds the project if necessary. Useful during development for real-time feedback.
 - *test*: Executes the test suite using Angular's testing framework (Karma and Jasmine by default).
 - *lint*: Runs ESLint to identify and report on patterns in JavaScript, helping to ensure code quality.
 - *lint:fix*: Runs ESLint with the option to automatically fix problems that can be corrected without user intervention.
- **private:** When set to true, prevents the package from being accidentally published on npm.
- **dependencies:** Lists all packages required for the application to run under normal conditions like Angular itself, RxJS, zone.js.
- **devDependencies:** Contains development-only packages like Angular CLI, TypeScript, and testing tools that are not needed in production.

Package-lock.json

This file is automatically generated when Node.js dependencies are installed (using `npm install`). It serves to lock down the exact versions of each installed package and its dependencies. This ensures that the project is consistent and predictably reproducible across different setups and deployments, avoiding issues that might arise from version mismatches of dependencies.

5.3.1.7. Application Entry and Global Resources

Index.html

The `index.html` file acts as the entry point of the Angular application. When the application is loaded in the browser, this is the first file that is served. It contains the root element `<app-root></app-root>` where the Angular app will be bootstrapped. The main chapters of this file are:

- **meta tags:** Essential for setting character set and viewport settings to enhance mobile accessibility.
- **link tags:** Include global stylesheets such as Bootstrap and a favicon.
- **app-root:** Placeholder directive where the Angular application attaches and renders.

Main.ts

This is the main entry file for the Angular application. It is where you instruct Angular to bootstrap your root application module (`AppComponent`), using Angular-specific bootstrapping tools. In this specific project, the file contains:

- **bootstrapApplication:** Function from Angular's platform-browser package used to bootstrap the application's root component.
- **AppComponent:** The root component that Angular first initializes.
- **appConfig:** Configuration used during bootstrapping.
- **Polyfills:** Imported to ensure browser compatibility.

Polyfill.ts

Polyfills allow web applications to maintain compatibility with different browsers, ensuring that features of JavaScript that are used in the app but not supported by older browsers work properly.

Styles.scss

This file is used for global styles of the app and can also import other style sheets. It is the main stylesheet where you can declare styles that should apply globally throughout the Angular application.

On this project, only the bootstrap CSS library is provided globally for styling the application.

5.3.1.8. Assets

In an Angular project, the assets folder is a special directory used to store static files like images, fonts, external libraries, stylesheets, and JavaScript files. Anything placed in this folder is copied as-is into the output directory (dist/ by default) during the build process. This makes the assets directory ideal for content that does not need to be processed by Webpack or any other build tool that Angular uses internally.

In the current project, the assets folder contains:

- **.png images:** images used throughout the application for various user interface components.
- **Styles subfolder:** contains CSS and JavaScript files related to Bootstrap.

5.3.2. Angular Application

5.3.2.1. Components

Main Component

The main component in an Angular application acts as the primary controller for the root template. This component orchestrates how the other components and views are displayed and managed within the application. For the Angular application described here, the main component's functionality is spread across several files, each serving a unique purpose in the application's architecture:

1. **App.component.ts:** This file is the Typescript file for AppComponent. It contains the logic for the component, including properties and metadata dictate how the component behaves. The code of the main component of the application described in this example can be structured by:
The component decorators configure aspects like the selector (app-root), template and styles. The standalone property indicates it does not require a module but imports what it needs directly.
The routerOutlet import is necessary for routing, allowing angular to insert components as routes change.
The title is a property that could be used for dynamic bindings in the template.
2. **App.component.html:** This file is the template for the AppComponent. It serves as the view layer, displaying the HTML content which users interact with directly.
3. It is structured in a way that uses bootstrap for layout, displaying a Siemens logo, an application title, a description and links to important resources. The <router-outlet> directive acts as a placeholder where the router can display different views/components based on the navigation state.
4. **App.component.scss:** specific styles that only apply onto the AppComponent. Being empty indicates no specific styles are overridden for this component.

Alarms

The Alarms Component is an integral part of the application designed to interact with Siemens S7 PLCs via the S7 Webserver API. It fulfills the role of displaying and managing alarm data, allowing users to view current alarms and acknowledge them. The component is split into a template (HTML), a style sheet (SCSS, although not used here), and the TypeScript logic.

The alarms.component.html is the template for the Alarms component. It provides the user interface for viewing and interacting with alarm data fetched from a PLC. This template displays key alarm data and provides a form for acknowledging alarms. It also provides functionality for updating the displayed data using Angular's data binding and structural directives.

The alarm.component.ts manages the data and behavior of the Alarms component. It interacts with the webserver API to fetch and manage alarm data. The code can be categorized by:

- **Initialization and Dependency Injection:** Uses Angular's dependency injection to get services and Angular's form building capabilities.
- **Fetch and Manage Alarms:** Implements methods using @siemens/simatic-s7-webserver-api to communicate with the Siemens Webserver API. This includes fetching alarm details and sending acknowledgments based on user interactions.
 - *RequestConfig and Filters:* Configuring requests to the API, including what data to ask for.
 - *acknowledgeAlarm:* Method to send acknowledgment back to the PLC concerning specific alarms.

Data View

The Data View Component displays time and data received from Siemens S7 PLCs. This component is designed to regularly poll the PLC for updated data values and timestamps, display them in a structured format, and handle any data communication errors gracefully. The main features of this component include setting up data polling, fetching data from the PLC, and formatting the received values for display.

The `dataView.component.html` provides the HTML structure for the DataView component which displays data in a tabular form updated every 2 seconds. The `*ngFor` directive repeatedly renders a row for each entry in the `dataStructArray`. It also includes Data formatting functions (`formatValue`) are used to ensure proper display of the data values.

The `dataView.component.ts` manages the lifecycle and data interactions for the DataView component, especially handling data fetching from a PLC and updating the view accordingly. The code can be structured:

- **Imports:** Necessary Angular modules for template-driven forms, routing, and common functionalities.
- **Lifecycle Hooks:** `OnInit` and `OnDestroy` to manage setting up and tearing down the interval-based data polling.
- Using services to create configuration settings necessary for communicating with the PLC securely.
- Setting up an `RxJS` interval to fetch data every 2 seconds, and cleanly unsubscribing during the component's destruction to prevent memory leaks.
- Data parsing for formatting the data received from the PLC into an array of data structures ready for user interface consumption.

The PLC interaction using the webserver API is handled by:

- **`createPlcProgramRead`:** Initializes a read operation based on the provided configuration.
- **`bulkExecute`:** Sends a bulk read request which is essential for efficient data retrieval, particularly when polling data frequently.

Diagnostic Buffer

The Diagnostic Buffer Component is designed to view diagnostic data from a Siemens S7 PLC. This includes handling data concerning various statuses and diagnostics of the system, which are useful for monitoring and troubleshooting purposes. It retrieves diagnostic entries, formats them for display, and provides functionality for refreshing the data manually.

The `diagnosticbuffer.component.html` serves as the user interface for displaying the data received from the PLC: It includes a UI for manually refreshing data and viewing detailed entries. The refresh button is wired to re-fetch and update the display data.

The `diagnosticBuffer.component.ts` handles fetching, formatting and refreshing diagnostic data from the PLC. In this case, the integration with the `@siemens/simatic-s7-webserver.api` is handled by the `fetchDiagnosticBuffer` function that configures the request with filtering options to include the necessary attributes and handles data formatting for the user interface. It also uses user session storage to manage the necessary PLC configuration data like token and address.

At the end there is also a function for data parsing and formatting to format each timestamp for human readability and incorporates it into the data structure used for displaying the entries, ensuring that the data is both comprehensive and accessible.

Landing Page

The Landing Page Component serves as the primary navigation hub and layout foundation for the Angular application. It organizes the various sub-components (like Overview Tank, Data View, Plant Status, etc.) into a cohesive interface with sidebar navigation. This component is crucial for providing a user-friendly and accessible way to navigate between different functionalities within the application.

The `landing-page.component.html` provides the HTML structure for the application's main navigation layout, integrating sidebar links with router functionality to switch views based on user interactions. The sidebar navigation utilizes Bootstrap's grid layout for responsive design. The sidebar contains links that are integrated with Angular's Router, facilitating single-page application behavior without page reloads. The router outlet acts as a placeholder that Angular dynamically fills based on the current router state. It renders the component associated with the active route.

The `landing-page.component.ts` configures the component's metadata and integrates routing functionality to enable navigation between different views within the application. The code can be split into:

- **Component Decorator:** defines the component's basic metadata. The templateUrl points to the HTML file, while the imports array includes RouterModule, necessary for enabling the routing functionalities within the template. The stand-alone property means that it doesn't require a module to function, but it can import other modules or components directly that it needs to function, in this case, RouterModule.

Log in

The Login Component facilitates user authentication into the application by interfacing with a Siemens PLC. It collects user credentials and PLC IP address, communicates with the backend for authentication, and handles navigation based on the authentication status.

The login.component.html provides the form interface for users to input their login credentials including the PLC IP address, username, and password. Each input field is bound to component properties (plcIpAddress, username, password) using [(ngModel)] for two-way data binding, which simplifies form handling. The form submission is handled by handleLogin method, triggered by the form's submit event.

The login.component.ts manages the login logic user authentication and navigation post-login. The code can be explained in a way that structures it:

- **Data Model Binding:** Integrates FormsModule which enables the use of [(ngModel)] in the template for two-way data binding.
- **Authentication and Routing Logic:**
 - The handleLogin function communicates with AuthService to perform the login and conditionally navigates to the landing page upon successful login.
 - It implements additional logic to handle periodic login, ensuring that the session remains active.

Overview Tank

The Overview Tank Component is designed to provide data visualization and control for processes related to a tank in an industrial setting. This component displays key metrics such as flow rate and tank levels, and it allows users to interact with the system by starting, stopping, and controlling valves directly from the interface.

The overview-tank.component.html displays the status of the tank including various tank level measurements and provides user controls for process operations. The template includes dynamic image sources that change based on system status to visually represent the tank's state. Interactive buttons allow users to directly control the process (e.g., starting, stopping, or adjusting valves), enhancing operational interactivity.

The overview-tank.component.ts manages data fetching for tank status and implements control logic to interact with the PLC for process operations. There are functions for initialization and monitoring that manage lifecycle events to initiate data fetching and continuous monitoring of tank conditions. Next, there are data fetching and state management functions to read data about various aspects of the tank system using the webserver API to gather process data efficiently.

At the end there are process control actions that thanks to the createPlcProgramWrite method, users can influence the running processes directly. Finally updates the system states upon successful operation and refreshes displayed data to reflect the changes immediately.

Syslog Buffer

The Syslog Buffer Component is designed to retrieve and display syslog data from a Siemens PLC system. It provides valuable insights into the system's operations by displaying logs that detail events, errors, and other system messages, essential for diagnostics and monitoring system health.

The syslogBuffer.component.html serves as the user interface to display the syslog entries fetched from the PLC along with statistics about the logs. For that uses a table that uses Angular's *ngFor directive to iterate over the array of syslog data, each displayed using a helper method getRawEntry() to ensure proper formatting.

The syslogBuffer.component.ts manages the logic for fetching syslog data from the PLC, interpreting it, and displaying it through the component template. The important methods here are:

- **Data Fetching:** fetchSyslogBuffer method utilizes the service for sending requests to the PLC and fetch syslog data.
- **Data Handling:** parses and checks the response to handle inconsistencies or gaps in data transmission.
- **Helper Function getRawEntry:** ensures that each entry is displayed correctly in the UI.

5.3.2.2. Services

A Service in Angular is a broad category encompassing any value, function, or feature that an application needs. A typical use of services is to share data or functionalities among components without duplicating code. Services are usually classes with a well-defined purpose and scope, particularly those decorated with `@Injectable()`, which marks them as available to be injected into components and other services.

The importance of services involves modularity and reusability of code, as well as maintainability. Furthermore, Angular's dependency injection framework allows you to manage dependencies across your components and services. It inserts needed dependencies rather than requiring them to be constructed in components, smoothing development and making the application easier to adapt and test.

Alarms Service

The Alarms Service is designed to interact with Siemens S7 PLCs to browse and acknowledge alarms. It leverages specific functionalities from `@siemens/simatic-s7-webserver-api` to create instances for browsing alarms and acknowledging them.

```
import { Injectable } from '@angular/core';
import { AlarmsBrowse, AlarmsAcknowledge, RequestConfig, Filters } from '@siemens/simatic-s7-webserver-api';

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

  createAlarmsBrowse(
    config: RequestConfig,
    authToken: string,
    language: string,
    count?: number,
    alarm_id?: number,
    filters?: Filters
  ): AlarmsBrowse {
    return new AlarmsBrowse(config, authToken, language, undefined, undefined, filters);
  }

  createAlarmsAcknowledge(
    config: RequestConfig,
    authToken: string,
    alarmId: string
  ): AlarmsAcknowledge {
    return new AlarmsAcknowledge(config, authToken, alarmId);
  }
}
```

- **Service Decoration (@Injectable):** The `@Injectable` decorator registers the service as a provider in the root injector. This setup implies that a single instance of `AlarmsService` will be created and shared across the application, promoting efficient memory usage and data consistency.
- **Method: createAlarmsBrowse:** Configures and returns an `AlarmsBrowse` instance which is used to fetch alarm information from a PLC. Parameters include configuration, authentication token, language, optional count, a specific alarm ID, and filter settings, allowing extensive customization of the fetch operation.
- **Method: createAlarmsAcknowledge:** Generates an `AlarmsAcknowledge` instance necessary to acknowledge alarms in the system. It takes the necessary configuration and authentication token, along with the specific `alarmId` to be acknowledged.

Auth Service

Auth Service is essential for managing authentication processes within applications that interface with Siemens S7 PLCs. It handles logging in to the system, managing session tokens, and periodically refreshing authentication to ensure continuity in user sessions.

The service setup is compounded by:

```
import { Injectable } from '@angular/core';
import { RequestConfig, ApiLogin } from '@siemens/simatic-s7-webserver-api';
import { RequestConfigService } from './request-config.service';

@Injectable({
  providedIn: 'root'
})
export class AuthService {
  private authToken: string | null = null;
  private config: RequestConfig;

  constructor(private requestConfigService: RequestConfigService) {
    this.config = this.requestConfigService.createConfig('https', false);
  }
}
```

- **@Injectable**: Marks the class as available for dependency injection, with the service scoped to the root so it is available application wide.
- **Configuration**: Initializes a RequestConfig object using RequestConfigService, setting the ground for all communications with the Webserver API, particularly for secure connections (https).

The methods available through the service are:

```
async loginToPLC(plcAddress: string, username: string, password: string): Promise<boolean> {
  ...
  const login = await new ApiLogin(this.config, username, password, false).execute();
  ...
}
```

- This method handles the login process using the ApiLogin class from the @siemens/simatic-s7-webserver-api. It configures the login with the given credentials and PLC address, executes it, and processes the result. If successful, it stores the received authentication token both locally and in session storage for later use.

```
startPeriodicLogin(): void {
  ...
  setInterval(async () => {
    const loginSuccess = await this.loginToPLC(plcAddress, username, password);
    if (!loginSuccess) {
      console.warn('Failed to refresh auth token.');
```

- Sets up a periodic execution of the login function to ensure that the session remains active. It refreshes the authentication token every minute.

```
getAuthToken(): string | null {
  ...
  return this.authToken || (storage ? storage.getItem('authToken') : null);
}
```

- Retrieves the current authentication token from either memory or session storage, providing a unified access point for consuming components or services to authenticate subsequent requests.

Diagnostic Buffer Service

The Diagnostic Buffer Service provides insights into the health and status of the system. It utilizes the Siemens Webserver API to access diagnostic buffer data, which includes detailed logs and system events that are crucial for monitoring and troubleshooting PLC operations. The service setup is compound of:

```
import { Injectable } from '@angular/core';
import { DiagnosticBufferBrowse, RequestConfig, Filters } from '@siemens/simatic-s7-webserver-api';

@Injectable({
  providedIn: 'root',
})
export class DiagnosticBufferService {
  createDiagnosticBufferBrowse(
    config: RequestConfig,
    authToken: string,
    language: string,
    count?: number,
    filters?: Filters
  ): DiagnosticBufferBrowse {
    return new DiagnosticBufferBrowse(config, authToken, language, count, filters);
  }
}
```

- **@Injectable:** Marks the class as one that participates in the dependency injection system. The `providedIn: 'root'` parameter ensures that a single instance of `DiagnosticBufferService` is created and available throughout the application.
- **Method: `createDiagnosticBufferBrowse`:** Configures and initiates a request to browse the diagnostic buffer of the PLC system.
 - Parameters:
 - *config*: The `RequestConfig` object containing the communication settings like protocol and address.
 - *authToken*: Authentication token required to authorize the request against the PLC.
 - *language*: The language parameter to localize the diagnostic messages (if localization is supported).
 - *count*: Optional count of how many entries to fetch.
 - *filters*: Optional `Filters` to apply to the fetching process, which helps in refining the diagnostic data retrieved.
 - **Return:** Instantiates and returns a `DiagnosticBufferBrowse` object configured with the necessary parameters which provide the functionality to execute the diagnostic buffer browsing operation.
- **Usage of `DiagnosticBufferBrowse`:** This created object is used to make an actual API call to the Siemens PLC and fetches the diagnostic buffer data. It encapsulates the necessary details and methods to communicate with the API, handle the response, and deal with any exceptions or errors during the communication.

Plc Program Service

The Plc Program Service is a critical part of the infrastructure that interacts with Siemens S7 PLCs for reading and writing program tags. This service facilitates direct manipulation and monitoring of PLC data values, essential for controlling and observing industrial processes directly from the application. The service setup includes:

```
import { Injectable } from '@angular/core';
import { PlcProgramRead, PlcProgramWrite, RequestConfig } from '@siemens/simatic-s7-webserver-api';

@Injectable({
  providedIn: 'root',
})
export class PlcProgramService {
  createPlcProgramRead(
    config: RequestConfig,
    authToken: string,
    Var: string,
    mode?: string
  ): PlcProgramRead {
    return new PlcProgramRead(config, authToken, Var, mode);
  }
  createPlcProgramWrite(
    config: RequestConfig,
    authToken: string,
    Var: string,
    value: unknown,
    mode?: string
  ): PlcProgramWrite {
    return new PlcProgramWrite(config, authToken, Var, value, mode);
  }
}
```

- **@Injectable:** Marks this service as eligible for dependency injection, with the `providedIn: 'root'` parameter ensuring it is available throughout the application as a singleton.
- **Method: `createPlcProgramRead`:** Sets up and returns a `PlcProgramRead` object, which is used to read tags from the PLC.
 - **Parameters:**
 - *config*: Configuration of the request including PLC address and protocol details.
 - *authToken*: Token for authenticating the request.
 - *Var*: The name of the PLC tag to be read.
 - *mode*: (Optional) The mode of reading, which could dictate how the data is read from the PLC.
 - **Operation:** This method initializes a `PlcProgramRead` instance equipped with all necessary parameters to perform a read operation.

- **Method: createPlcProgramWrite:** Configures and returns a PlcProgramWrite object, which allows writing values to the PLC.
 - Parameters:
 - *config*: Contains information required to connect and authenticate with the PLC.
 - *authToken*: Authentication token necessary to validate the request.
 - *Var*: The tag in the PLC to write to.
 - *value*: The value to write to the specified tag.
 - *mode*: (Optional) Specifies how the write operation will be handled by the API.
 - *Operation*: This method creates a PlcProgramWrite instance that can execute the write operation using the provided details.

Request Config Service

The Request Config Service is a fundamental component in setting up interactions with Web API of Siemens PLCs by systematically creating and configuring RequestConfig objects. These objects hold the necessary details for connecting to and communicating with a PLC securely and effectively. The service setup is:

```
import { Injectable } from '@angular/core';
import { RequestConfig } from '@siemens/simatic-s7-webserver-api'; // Adjust the import path

@Injectable({
  providedIn: 'root', // This ensures it's provided globally
})
export class RequestConfigService {
  constructor() {}

  createConfig(protocol: string, verifyTls: boolean): RequestConfig {
    const config = new RequestConfig();

    config.protocol = protocol;
    config.verifyTls = verifyTls;
    return config;
  }
}
```

- **@Injectable:** Marks this class as a service that can be injected, ensuring it is available throughout the application due to the providedIn: 'root' configuration. This means the service is a singleton and only one instance exists during the lifecycle of the application.
- **createConfig Method:** Provides a consistent method to configure and instantiate RequestConfig objects used for PLC interactions.
 - Parameters:
 - *protocol*: The communication protocol to use, typically "https" for secure connections.
 - *verifyTls*: A Boolean that specifies whether the connection should verify TLS/SSL certificates, an important security consideration for protecting data integrity and confidentiality.
 - **Operation:** The method initializes a RequestConfig object, sets its protocol and verifyTls properties based on parameters, and returns this configured object.

Syslog Buffer Service

The SyslogService provides a specialized interaction with syslog data from the Web API of Siemens S7 PLCs. It streamlines the retrieval of log entries, which are crucial for monitoring operational processes, troubleshooting issues, and ensuring the system's integrity. The service setup:

```
import { Injectable } from '@angular/core';
import { SyslogBrowse, RequestConfig } from '@siemens/simatic-s7-webserver-api';

@Injectable({
  providedIn: 'root',
})
export class SyslogService {
  createSyslogBrowse(config: RequestConfig, authToken: string): SyslogBrowse {
    return new SyslogBrowse(config, authToken);
  }
}
```

- **@Injectable:** Marks the class for dependency injection with Angular's injector, specifying `providedIn: 'root'` to make it a singleton. This ensures it is available globally and only a single instance of the service is created throughout the lifecycle of the application.
- **Method: `createSyslogBrowse`:** Initializes a `SyslogBrowse` instance to facilitate fetching syslog data from a PLC using the `@siemens/simatic-s7-webserver-api`.
 - **Parameters:**
 - `config`: A `RequestConfig` object that encapsulates the configuration needed for the API calls, such as the protocol and the PLC's URL.
 - `authToken`: A string representing the authentication token required to authorize the API call.
 - **Operation:** This method constructs and returns an instance of `SyslogBrowse`, prepared with all the necessary configuration and credentials. This object provides the functionality required to perform and handle syslog data retrieval operations.

5.3.2.3. Configuration

In Angular, the `app.config.ts` file typically serves as a configuration file that sets up application-wide settings or providers that are central to the functionality of the application. This file is part of Angular's dependency injection system, which allows you to configure providers for services, routes, and other necessary setup at the application's root level. Using `app.config.ts` aids in maintaining a clean and modular architecture by separating configuration concerns from business logic and component definitions.

The primary use of `app.config.ts` is to provide and configure services, routing, and other core functionalities. It helps in:

- Centralizing the configuration for services and routing which improves manageability and scalability.
- Setting up providers that can be injected into any part of the application, ensuring a consistent configuration throughout.
- Configuring global error handlers, routing strategies, and HTTP configurations that are essential for application performance and behavior.

The app.config.ts from this project contains:

```
import {ApplicationConfig } from '@angular/core';
import {provideRouter, withHashLocation, NavigationError, withNavigationErrorHandler } from
'@angular/router';
import {provideHttpClient } from '@angular/common/http';
import {routes } from './app.routes';
import {AuthService } from './services/auth.service';
import {RequestConfigService } from './services/request-config.service';
import {AlarmsService } from './services/alarms.service';
import {SyslogService } from './services/syslogbuffer.service';
import {PlcProgramService } from './services/plcprogram.service';
import {DiagnosticBufferService } from './services/diagnosticbuffer.service';
export const appConfig: ApplicationConfig = {
  providers: [
    provideRouter(
      routes,
      withHashLocation(),
      withNavigationErrorHandler((error: NavigationError) => {
        console.error('Navigation Error:', {
          url: error.url,
          error: error.error
        });
        return true; // Prevent default error handling
      })
    ),
    provideHttpClient(),
    // Services
    AuthService,
    RequestConfigService,
    AlarmsService,
    SyslogService,
    PlcProgramService,
    DiagnosticBufferService
  ]
};
```

- **Routing Configuration:** provideRouter is used to configure the application's routing. This setup includes routes from app.routes.ts, hash-based location strategy with withHashLocation(), and custom navigation error handling using withNavigationErrorHandler(). This error handler logs navigation errors and can be extended to include user redirection or additional error resolution strategies.
- **HTTP Client Configuration:** provideHttpClient() sets up the HTTP client for use throughout the application, enabling HTTP communications with backend services.
- **Service Providers:** List of all the services that should be provided application wide. This includes:
 - AuthService: Handles authentication-related functionality.
 - RequestConfigService: Provides configurations for HTTP and API interactions.
 - AlarmsService: Manages interaction with the alarms systems of the PLC.
 - SyslogService: Deals with fetching and handling syslog data.
 - PlcProgramService: Facilitates reading from and writing to PLC programs.
 - DiagnosticBufferService: Manages interactions with the diagnostic buffer of the PLC.

5.3.2.4. Routing

The `app.routes.ts` file defines the routes for an Angular application. These routes map navigation paths to components, enabling the application to display different views to the user based on the URL they access. Angular uses this configuration to handle routing internally, operations which include loading the appropriate component when a path is navigated to, and handling any data associated with the route.

The primary use of `app.routes.ts` is to configure the routes of the application in a centralized manner:

1. **Organization and Navigation:** It organizes the application's navigation structure, defining which component should be loaded for each path and managing any nested routes (child routes).
2. **Accessibility:** It controls the accessibility of components based on the navigation paths, often including route guards for certain paths that require authentication or authorization.
3. **Modularity:** By separating routes into a specific file or module, Angular applications can maintain clean separation of routing concerns from feature and business logic, which helps in large-scale application development.

```
import {Routes } from '@angular/router';
import {LoginComponent } from '../components/login/login.component';
import {LandingPageComponent } from '../components/landing-page/landing-page.component';
import {OverviewTankComponent } from '../components/overviewTank/overview-tank.component';
import {DataViewComponent } from '../components/dataView/dataView.component';
import {PlantStatusComponent } from '../components/plantStatus/plantStatus.component';
import {DiagnosticBufferComponent } from
'../components/diagnosticBuffer/diagnosticBuffer.component';
import {SyslogBufferComponent } from '../components/syslogBuffer/syslogBuffer.component';
import {AlarmsComponent } from '../components/alarms/alarms.component';
export const routes: Routes = [
  { path: '', redirectTo: '/login', pathMatch: 'full' },
  { path: 'login', component: LoginComponent },
  { path: 'landing',
    component: LandingPageComponent,
    children: [
      { path: '', redirectTo: 'overview-tank', pathMatch: 'full' },
      { path: 'overview-tank', component: OverviewTankComponent },
      { path: 'data-view', component: DataViewComponent },
      { path: 'plant-status', component: PlantStatusComponent },
      { path: 'diagnostic-buffer', component: DiagnosticBufferComponent },
      { path: 'syslog-buffer', component: SyslogBufferComponent },
      { path: 'alarms', component: AlarmsComponent }
    ]
  }
];
```

- **Root and Login Routes:** Includes a root route that redirects to `/login`, ensuring that any access to the base URL navigates users to the login view.
- **Landing Page Route:** Acts as a parent route for various child routes that make up different functional parts of the application, such as overview of tanks, data views, plant status, and more.
- **Child Routes:** Nested within the 'landing' route, these paths lead to specific functionalities and components of the application, handling different aspects like alarms and system diagnostics.

5.3.2.5. Angular Development Server

The Angular development server is a built-in local web server that comes with the Angular CLI (Command Line Interface). It is fundamentally designed to aid in the rapid development and testing of Angular applications by providing a simple and efficient means to serve and reload your application as you make changes to the codebase.

Core Functions

1. **Serve Application:** The primary role of the Angular development server is to serve the Angular application to a browser. It does so by compiling the application into deployable JavaScript, HTML, and CSS files and hosting these on a local server typically accessible via `http://localhost:4200`.
2. **Live Reloading:** One of the most significant features of the Angular development server is its ability to watch for changes in files and automatically recompile the modified parts of the application. It then refreshes the browser to reflect these changes, giving developers immediate feedback on updates. This feature enhances productivity by eliminating the need to manually restart the server or refresh the page to see changes.
3. **Hot Module Replacement (HMR):** Beyond simple live reloading, the Angular development server can also support Hot Module Replacement. HMR improves the live-reload capability by updating modules in the browser in real time without needing to refresh the whole page, preserving the current state of the application which can be very useful for maintaining the application's state especially during styling and interaction tweaks.

How It Operates

1. **Build and Compilation:** When execute commands like `ng serve`, the development server uses Webpack to compile the TypeScript, HTML, and CSS files of the Angular application into JavaScript files that the browser can execute. This compilation includes the Angular framework, application code, third-party libraries, and assets.
2. **Serving Assets:** The Angular development server also serves static assets from the `assets` directory and other configured paths. These assets are usually images, fonts, and static files necessary for the application.
3. **Advanced Configuration with `angular.json`:** The `angular.json` config file in an Angular project specifies a wide range of build, serve, and watch options for the development server. These configurations allow fine-tuning behaviors like the default HTTP port, enabling HTTPS, proxying API requests for backend integration.
4. **Debugging Support:** The server also simplifies debugging by providing source maps, which are files that map from the transformed source to the original source, enabling you to see the original TypeScript code directly in the browser's developer tools.
5. **Proxying API Requests:** For applications that need to interact with backend servers (which might be on different domains, especially during development), the Angular development server can be configured to proxy certain API requests to another server to work around CORS (Cross-Origin Resource Sharing) policies.

How to run the Angular Development Server for this application

Open the terminal in the project directory and type the command: `ng serve`

The following message will appear on the terminal when the server is running:

```
** Angular Live Development Server is listening on localhost:4200, open your browser on http://localhost:4200/~myApp
**
```


6. Appendix

6.1. Service and support

SiePortal

The integrated platform for product selection, purchasing and support - and connection of Industry Mall and Online support. The SiePortal home page replaces the previous home pages of the Industry Mall and the Online Support Portal (SIOS) and combines them.

- **Products & Services**
In Products & Services, you can find all our offerings as previously available in Mall Catalog.
- **Support**
In Support, you can find all information helpful for resolving technical issues with our products.
- **mySiePortal**
mySiePortal collects all your personal data and processes, from your account to current orders, service requests and more. You can only see the full range of functions here after you have logged in.

You can access SiePortal via this address: sieportal.siemens.com

Industry Online Support

Industry Online Support is the previous address for information on our products, solutions and services.

Product information, manuals, downloads, FAQs and application examples - all information is available with just a few mouse clicks: support.industry.siemens.com

Technical Support

The Technical Support of Siemens Industry provides you fast and competent support regarding all technical queries with numerous tailor-made offers – ranging from basic support to individual support contracts.

Please send queries to Technical Support via Web form: support.industry.siemens.com/cs/my/src

SITRAIN – Digital Industry Academy

We support you with our globally available training courses for industry with practical experience, innovative learning methods and a concept that's tailored to the customer's specific needs.

For more information on our offered trainings and courses, as well as their locations and dates, refer to our web page: siemens.com/sitrain

Industry Online Support app

You will receive optimum support wherever you are with the "Industry Online Support" app. The app is available for iOS and Android:



6.2. Links and literature

Nr.	Thema
\1\	Siemens Industry Online Support https://support.industry.siemens.com
\2\	Link to this entry page of this application example https://support.industry.siemens.com/cs/ww/en/view/68011496
\3\	HTML5, JavaScript https://www.w3schools.com/html/ https://www.w3schools.com/js/
\4\	SIMATIC S7-1200 G2 Programmable Controller, System Manual https://support.industry.siemens.com/cs/ww/en/view/109972011
\5\	S7-1500 Web server, Function Manual https://support.industry.siemens.com/cs/ww/en/view/109977246
\6\	SIMATIC S7-1500, System Manual https://support.industry.siemens.com/cs/ww/en/view/59191792
\7\	SIMATIC STEP 7 Basic/Professional V20 and SIMATIC WinCC V20 https://docs.tia.siemens.cloud/
\8\	HTML and CSS, Practical Formulas for Beginners Robert R. Aguilar Mitp ISBN 978-3-8266-1779-9
\9\	HTML manual Stefan Münz / Wolfgang Nefzger Franzis Verlag (Publishing house) ISBN 3-7723-6654-6
\10\	JavaScript and Ajax, the comprehensive manual Christian Wenz Galileo Press ISBN 978-3-8362-1128-4

Table 6-1

6.3. Change documentation

Version	Date	Modification
V1.0	02/2014	First version
V2.0	06/2015	The applications "Creating and Using User-Defined Web Pages for S7-1200" and "Creating and Using User-Defined Web Pages for S7-1500" were merged.
V2.1	10/2015	Correction
V2.2	10/2016	The application example is programmed exclusively with HTML5.
V3.0	09/2019	Update TIA Portal V15.1 (v3.0.1: project user added)
V4.0	03/2020	Extension of the application example with <ul style="list-style-type: none">HTML pages with JavaScriptHTML pages with Web API and JavaScript
V5.0	05/2025	<ul style="list-style-type: none">Extension of the application example to SIMATIC S7-1200 G2Remove of AWP commands in HTML pagesExclusive HTML Pages with Web API, Typescript and client packageAngular framework integration

Table 6-2