D6.1 – Reference Architecture of the Assurance Framework Toolkit

Security Assurance Framework for Networked Vehicular Technology

Abstract

SAFERtec proposes a flexible and efficient assurance framework for security and trustworthiness of Connected Vehicles and Vehicle-to-I (V2I) communications aiming at improving the cyber-physical security ecosystem of "connected vehicles" in Europe. The project will deliver innovative techniques, development methods and testing models for efficient assurance of security, safety and data privacy of ICT related to Connected Vehicles and V2I systems, with increased connectivity of automotive ICT systems, consumer electronics technologies and telematics, services and integration with 3rd party components and applications. The cornerstone of SAFERtec is to make assurance of security, safety and privacy aspects for Connected Vehicles, measurable, visible and controllable by stakeholders and thus enhancing confidence and trust in Connected Vehicles.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no 732319

DX.X & Title:	D6.1 – Reference Architecture of the Assurance Framework Toolkit
Work package:	WP6
Task:	T6.1 Reference Architecture of the Assurance Framework Toolkit (AFT)
Due Date:	30 April 2019
Dissemination Level:	PU
Deliverable Type:	R

Authoring and review process information	
EDITOR Angelos Stamou/ICCS	**DATE** 10-09-2019 21-11-2019 17 -07-2020
CONTRIBUTORS Panagiotis Pantazopoulos/ICCS	**DATE** 25-11-2019 08-07-2020 23-07-2020
REVIEWED BY Silvia Capato/Swarco Kostas Maliatsos/UPRC	**DATE** 04-12-2019 09-12-2019
LEGAL & ETHICAL ISSUES COMMITTEE REVIEW REQUIRED?	
NO	

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no 732319

Document/Revision history

Version	Date	Partner	Description
V0.1	07/01/2019	ICCS	Table of Contents - First draft
V0.2	07/02/2019	ICCS	Inputs to Section 3
V0.3	11/03/2019	ICCS	Updates of Section 4 structure, formatting, inputs to Section 1
V0.4	20/04/2019	ICCS	Edits in Sections 2 and 4, addition of figures
V0.5	04/05/2019	ICCS	Update of sub-sections under section 4
V0.6	10/09/2019	ICCS	Update of the architecture to match the SAFERtec assurance framework
V0.7	25/11/2019	ICCS	Update of the architecture in line with comments (from the consortium experts) on an online draft-instance of the toolkit
V0.8	04/12/2019	Swarco	Internal review comments
V0.9	09/12/2019	UPRC	Internal review comments
V1.0	11/12/2019	ICCS	Final version
V1.1	27/07/2020	ICCS	Revised to meet the review comments. The updates amount to: • Updates to highlight how the SAFERtec framework work and the relevant updates (compared to the original plan) affect the AFT design (Section 1) • Basic roles and functionality have been more clearly described (Section 2.1, Figure 1) • Subsection 2.4 presents a systematic description of the proposed architecture in line with the Rational Unified Process[1] methodology. Relevant references added. • The new Section 6 clarifies some welcome AFT features (interoperability, modularity, extensibility and computational efficiency)

[1] https://www.sciencedirect.com/topics/computer-science/rational-unified-process

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no 732319

Table of Contents

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no 732319

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no 732319

Table of Figures

List of Tables

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no 732319

Acronyms and abbreviations

Abbreviation	Description
AFT	Assurance Framework Toolkit
AOP	OPerational Assurance (evaluation class)
API	Application Programming Interface
AJAX	Asynchronous JavaScript and XML
ANSI	American National Standards Institute
CC	Common Criteria
CVS	Connected Vehicle System
DOM	Document Object Model
ECMA	European Computer Manufacturers Association
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IETF	Internet Engineering Task Force
IDE	Integrated development environment
ISO	International Organization for Standardization
JWT	JSON Web Token
MVC	Model–View–Controller
NPM	Node Package Manager
ORDBMS	Object-relational Database Management System
PP	Protection profile
RUP	Rational Unified Process
SFR	Security Functional Requirement
SPA	Single Page Application
ST	Security Target
SW	Software
ToE	Target of Evaluation
TSFI	TOE Security Function Interface

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no 732319

UI	User Interface
URL	Uniform Resource Locator
V2I	Vehicle-to-Infrastructure
VB	Visual Basic
W3C	World Wide Web Consortium
WWW	World Wide Web
XML	Extensible Markup Language

Table 1: List of Abbreviations

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no 732319

Executive Summary

This document presents the reference architecture of the SAFERtec Assurance Framework Toolkit (AFT). First, it goes through a comprehensive analysis of the involved technical requirements that relate to a broad set of needs. Essentially, the proposed architecture (described also from a Rational Unified Process standpoint [13]) can support the implementation of a toolkit that would be used to help with the development of an automotive product by supporting (in line with the SAFERtec Assurance Framework) the definition of the Security Profile/Target [1] for that product (ToE).

The derived architecture retains a certain level of generality to support quick adaption to the Security Assurance Framework (expected) outcome and at the same time serve as the basis for the software development of the toolkit. Certain updates of the architecture have been undertaken to align with the feedback received by considering a skeleton instance of the architecture[2].

The proposed architecture designed in line with well-known software standards will lay the ground to the implementation and testing of the SAFERtec AFT that facilitates the efficient introduction of security assurance arguments in the V2X automotive setting.

[2] This has been developed and released for testing purposes under the URL : https://safertec.iccs.gr/

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no 732319

1 Introduction

One of the most interesting outputs of SAFERtec amounts to the Assurance Framework Toolkit (AFT). The toolkit aims to implement the basic principles of the SAFERtec security assurance framework [2] i.e., to "transform" the structured way of obtaining security guarantees in a V2I setting into a robust and easy-to-use WWW toolkit. Its design principles are presented herein.

The document adopts a top-down approach. It first identifies some high-level technology-independent requirements (that point to the toolkit's technology needs) and then describes the general AFT functionality. The document sheds some further insights on the proposed design by relying on the Rational Unified Process (RUP) to prescribe the intended reference architecture (pointing to relevant best practices) [13].

A broad set of technical requirements are considered in order to fulfill the aforementioned needs. Subsequently, the document details the employed software tools along with the motivations behind each choice with the AFT front- and back- end parts considered separately.

The AFT architecture as well as the subsequent implementation task adhere to a number of well-established software standards:

- HyperText Markup Language (IETF, ISO, W3C) or HTTP, is a language which the language in which web pages are authored. Compliance with them ensures cross-browser compatibility.
- ECMAScript (ECMA, ISO) is the standard javascript specification [3] which will help make the toolkit's dynamic content compatible.
- Cascading Style Sheets (CSS) is a language specification [4] which provides browsers a standardized way to present content.
- JavaScript Object Notation (IETF, ECMA) is a widely-used standard way [5] to exchange textual information across systems and heavily used in AJAX applications.
- C# Language Specification (ECMA, ISO) is the specification for the framework language which will be used on the server-side component of the application.
- Structured Query Language (ANSI, ISO) is the language that most traditional databases implement as a means of interaction.

The document showcases the architecture along with the major modules, while providing insights on the corresponding interfaces. As the SAFERtec Assurance Framework is currently under update relying on on-going testing activities, emphasis is placed on the high-level software design (description) which can later be shaped to fit any possible changes. This provides the opportunity to make implementation decisions at a later stage while staying in line with the (intended) final framework output.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no 732319

SAFERtec has chosen to build its proposal on-top of the most credible and Internationally recognised security assurance standard, the Common Criteria (ISO/IEC 15408) [1]. Several reasons justify this decision, the highest possible level of the achievable assurance as well as its recognised credibility being some of the most important ones; importantly, those reasons are becoming (pointing to an appropriate SAFERtec decision) in view of ever-increasing automation level of connected vehicles.

The considered standard and accordingly the SAFERtec proposal (introduced in WP3) suggest a rich spectrum of evaluation tasks over the evaluated system (i.e., Target of Evaluation). SAFERtec seeks to considerably enhance the Common Criteria (CC) introducing a tailored-made framework of *direct* and *cost-efficient* applicability to the connected vehicles paradigm. The Common Criteria processes are therefore respected; for instance, the assurance level of any evaluation is not 'computable' (i.e. cannot be software-generated) but predefined by the standard, essentially determined by the depth of the evaluation tasks the standard suggests. This renders the functionality of the (originally envisioned) inference engine, not relevant. At the same time, carefully designed contributions aim to ensure the SAFERtec cost-efficiency: an innovative risk analysis to drive the requirements elicitation, proposals of additional evaluation classes (for devising system-level evaluation arguments), a dedicated knowledge base fed by Connected Vehicle Protection Profiles and finally, custom tools for supporting in an online-fashion the gathering/management of SAFERtec (or CC) evaluation data. AFT has been designed to serve this last purpose.

Along this line, the AFT design includes the necessary online functionality to facilitate/ease the gathering, organization, management of SAFERtec (or CC) evaluation data as well as the efficient compilation of the relevant outputs (e.g., Security Targets, Architecture specifications of the ToE). As such, all AFT software modules (e.g., knowledge base) have been designed (and subsequently developed) to meet the herein presented requirements.

1.1 Purpose of the Document

The document seeks to describe the reference architecture of the SAFERtec Assurance Framework Toolkit. This work will serve as the basis for the AFT implementation that follows.

1.2 Intended readership

Besides the project reviewers, this deliverable is addressed to any interested reader as it is of Public dissemination level.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no 732319

1.3 Inputs from other projects

No input from other projects was considered during the compilation of this deliverable.

1.4 Relationship with other SAFERtec deliverables

There is no direct dependency of the content included in this document on past SAFERtec deliverables. However, the architecture and the relevant SW technologies presented here will guide the implementation effort which will be reported in subsequent WP6 SAFERtec deliverables. Finally, the results of WP3 (mainly those that appear in D3.2) so far have been considered in the introduced AFT architecture.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no 732319

2 Conceptual Architecture of the AFT

SAFERtec develops and provides an Assurance Framework Toolkit (AFT) for Connected Vehicles, which will be realized as a modular software platform. AFT is enabled to host the identified CVS assets, the result of the SAFERtec risk analysis (expressed as threats, assumptions and security objectives) and the corresponding security functional requirements.

In a nutshell, the toolkit will mainly enable a structured/automated way (to help the product developer) compose an ST (PP) for an automotive ToE and address the issues imposed by the ST (PP). Those structured forms of (security assurance) evaluation data will be requested by a relevant CC evaluator and would otherwise come at a high time/resources cost for the ToE developer. AFT seeks to facilitate through its online functionality the cost-efficient compilation of the required data. Further extensions regarding the support of other security evaluation tasks are discussed in the Appendix A - Potential Extensions.

Following the Rational Unified Process [13] we provide the presentation of the AFT (software design and) reference architecture under a number of abstractions infusing user roles, best practices, standards and available (technology) tools. A software architecture through (concurrent) abstractions called views, each of which addresses a specific set of concerns.

2.1 Uses Cases View

Currently the Toolkit has been designed and implemented to support a basic set of use-cases, as shown in the following diagram (Figure 1). The leftmost ellipsis shapes refer to the main use-case of each role that are further broken down to details on the rightmost set.

Later in this document, the careful design and generality of the involved modules that effectively enable the extensibility of the current architecture will be shown.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no 732319

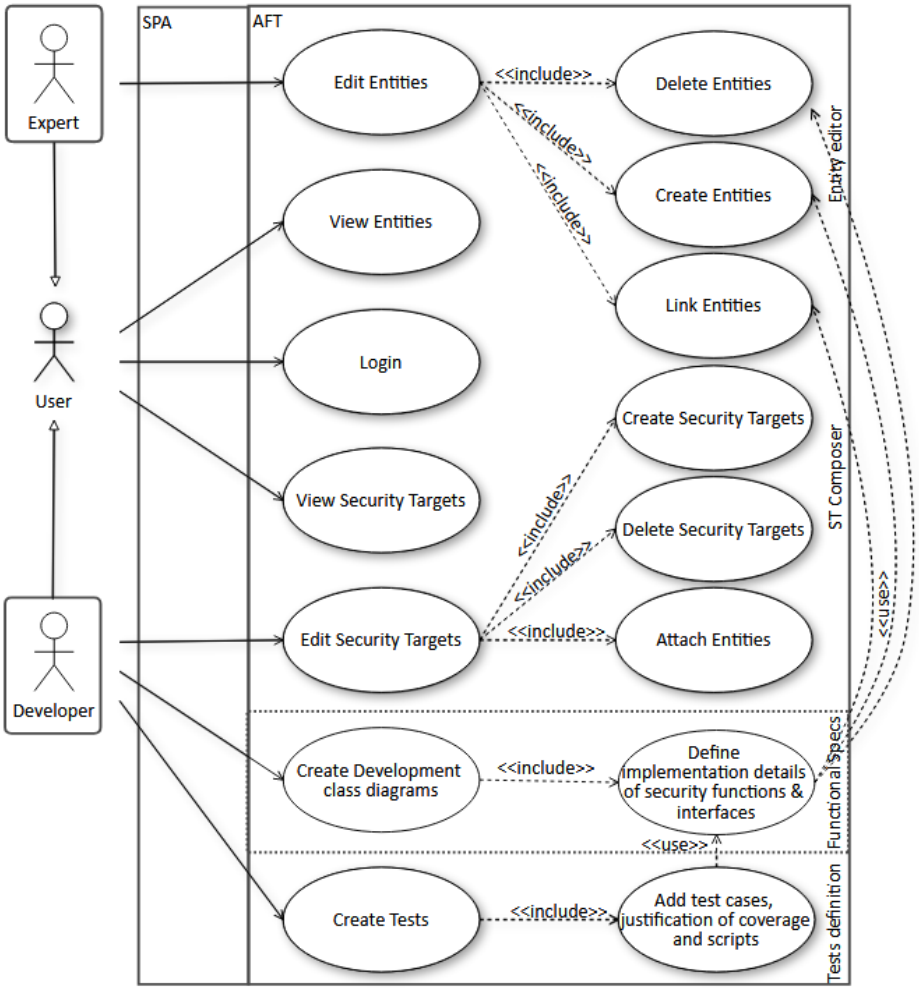


Figure 1: Basic AFT roles and supported functionality (Use Cases) of the toolkit

Each user role is linked to the main use-cases through plain arrows. Further dotted links point to the detailed functionality of each use-case. Two tags on the links have been used to characterize the supported functionality: “include” marks the scope of each use-case and “use” point to the usage of some functionality related to another use-case.

Next, we discuss the two identified roles of the AFT user (shown in the left part of Figure 1) and the expected functionality that the AFT architecture implements.

2.1.1 User Roles

AFT has been designed with two roles of users: one to recognize and support the certified expert user allowing for full access to the (internal parts of the)’ AFT. The second one corresponds to the

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement no 732319

typical user and allow him to operate the toolkit, retrieve information and prepare the evaluation input data required for security assurance evaluation of automotive products. These two share a common set of actions like logging in to the application, searching and viewing the data.

- **Expert**. The expert user is the one who can define entities and relations in the Toolkit and make them available to developers that constitute the 'common' users of the AFT.
- **Developer**. The developer user is the main (typical) audience of the Toolkit. He will rely on the AFT operation in order to use/exploit the information curated by the expert; thus, he will be assisted to *efficiently*[3] introduce security evaluation criteria for his automotive product (ToE) in line with the SAFERtec framework.

2.1.2 Use Cases

In what follows we provide a brief explanation of the use cases presented in the diagram of Figure 1.

- **Login.** The user is authenticated using a username and a password. This allows for subsequent authorization.
- **View Entities.** The user can browse previously entered entities, filter them by category and/or partial text.
- **View Security Targets.** The user can browse previously entered security targets and filter them by partial text.
- **Edit Entities.** The expert user can edit entities. This includes attributes like title, description and category. The user can also create and delete entities as well as link them together according to their type.
- **Edit Security Targets.** The expert user can edit security targets. This includes all sections of the security target as defined by the Common Criteria standard. The expert user can also create and delete security targets as well as attach entities to the security target depending on the security target product.
- **[extension to] Create Development Class Diagrams.** The developer can use a graphical environment to provide the ToE details in terms of functional specifications required for the ADV evaluation class.
- **[extension to] Create Tests.** The developer can rely on the AFT entities to define test cases in the context of the ATE evaluation class.

As AFT can be extended (see Appendix A - Potential Extensions), the above list will be eventually extended i.e., see the last two bullets, to cover the new functionality.

[3] The efficiency of the SAFERtec approach (compared to the typical CC approach) both in terms of time and cost will be comprehensively evaluated in the WP5 deliverables.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no 732319

2.2 Main modules

The toolkit consists of two main functionalities represented by two conceptual modules *i.e.*, the editor and the composer module.

2.2.1 Editor

The first one allows the expert user to define the involved entities (e.g., a security assumption relevant for the considered product/ToE) with as much detail as required. Those entities are enabled to correspond to the notions described in the Common Criteria framework [1].

2.2.2 Composer

The composer allows the product developer to browse the entities created by the expert user and utilize them in order to be assisted in a number of security evaluation tasks (*e.g.*, compose a Security Target for an automotive product of interest).

2.3 Interfaces

2.3.1 Single Page Application

The users may interact with the Toolkit relying on one interface, the Single Page Application. This provides a unified access to the application. The advantages of the approach are detailed in Section 4.2.

2.4 Description under a Logical View

In what follows we elaborate the details of a Logical View description [13] identifying various involved layers. For each layer we briefly discuss the relevant technical choices (e.g., employed SW services), standards and tools. We adopt an easy-to-follow presentation using tables and provide further references where needed.

2.4.1 User Interface Layer

Area	Products/Services/Components
Layout	No formal guidelines but the toolkit UI should follow a responsive and clean design.
Usability	No formal guidelines but the toolkit should be simple and prompt to user interaction.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no 732319

Standards	Valid HTML, CSS and Javascript must be produced which should render correctly on all major up-to-date desktop browsers.
Construction Tools	
1. Languages	1. Typescript for UI logic 2. HTML for layout structure 3. CSS for presentation
2. IDE	Visual Studio Community Edition 2017/2019
3. Information Stream	1. HTML, CSS and Javascript for initial loading of app logic and data 2. JSON for subsequent data loading as well as interaction
4. Design Strategies	Standard Angular MVVM project structure.
5. Package Manager	npm [10]
Components	
1. Authorization	JWT included in the requests
2. Transpiler	TSC compiler

2.4.2 Application Logic Layer

Components	
1. Languages	1. C# (.NET Core) [11] 2. SQL (auto generated)
2. IDE	Visual Studio Community Edition 2017/2019
3. Package Manager	NuGet
Service Components	
1. Compression	GZIP
2. Database Access	EF Core
3. Security	1. HTTPS for transport 2. HSTS policy
Pattern Usage for specific scenarios	
The application interface requires organizing into resources in accordance with REST practices	Model-View-Controller pattern or MVC allows for a clean design where the resources are grouped by class and the actions by method.
Software components must be modular and some unit testing will be required	Dependency Injection allows for testing by providing resources on class instantiation
Separation is required between instances of the application servicing different requests	Scoped Services initialize resources per request ensuring efficient use and separation of concerns
Some resources are not constantly required	Singleton Services are ideal for providing functionality to the application as a whole

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no 732319

2.4.3 Middleware Layer

Area	Products/Services/Components
Reverse Proxy	Nginx [15]
Application Server	Kestrel [16]
Runtime	Common Language Runtime

2.4.4 System Software Layer

Area	Products/Services/Components
Operating Systems	Linux
RDBMS	PostgreSQL [17]
Package Management	1. NuGet 2. npm [10]

3 Technical Requirements

Technical requirements, also known as non-functional requirements, describe the behaviour of the toolkit while AFT performs its functions. Those requirements need to be met in most of complex software systems [6] in order to ensure a number of welcome characteristics. A certain set of them, discussed below, are even more relevant for AFT in order to appropriately reproduce the SAFERtec framework.

3.1 Adaptability

The Framework is subject to change especially since cyber-security is a moving target. The AFT design should anticipate that and be able to accommodate any modifications with minimal cost. This can be achieved by using a modular architecture and enforcing clean abstractions. This way, clear interfaces are easily implemented in new components and fitted in an existing system. Towards the same end, any rigid and hardcoded functionality should be avoided.

3.2 Auditability

Obscurity in complex systems is all too common. Keeping operations easy to trace helps keep the project compliant with requirements, helps detect flaws in design and operation, supports security

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no 732319

and provides accountability. The system should support logging to ensure traceability and transparency without sacrificing performance. Obviously, sane data retention principles (needed to comply with Member State/EU regulations or recover business data in case of accidents, see [7]) must be followed with regards to logging.

3.3 Backup

Backup provisions are a prominent consideration of almost every software system and come with a straightforward rationale. The Toolkit is no different in this sense. Reliable and robust backup mechanisms must be in place and provide the means for disaster recovery.

3.4 Deployment

Easy software deployment is extremely useful yet somewhat overlooked. A good deployment mechanism allows for a short development to deployment cycle which brings a number of benefits such as timely updates, fast bug fixing, easy scaling and rapid recovery. For these reasons the Toolkit should be easily deployed with minimal downtime.

3.5 Documentation

Software documentation is also an essential component of every system. However, reliable, concise and up-to-date documentation is difficult and time consuming to prepare and maintain. Given how pivotal it is to ensure correctness, proper maintenance and untroubled development, it should be self-evident that good documentation must accompany the Toolkit.

3.6 Extensibility

New requirements are commonplace in involved systems. Oftentimes they are difficult to become apparent before the actual development starts. To compensate for this eventuality, the Toolkit must be extensible in order to incorporate new functionality in its existing instance. Modularity also helps along the same lines by providing clean starting points to add new components.

3.7 Interoperability

The Toolkit needs to be enabled to expose all included functionality to third party systems. A well-defined and documented API is essential for this. The corresponding interface should be easy to understand and develop in order to encourage its integration with other systems.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no 732319

3.8 Maintainability

The (single) greatest cost in a software product's lifecycle comes from maintenance. The Toolkit design must be easy to understand, correct and adapt, if need be. This reduces the complexity of the maintenance task as well as the time required.

3.9 Performance

Performance is important since sluggish systems tend to go unused and are a source of frustration to users. Thresholds are difficult to define with regards to both time and concurrency and are heavily dependent on the type, purpose and intended audience of the system. Care must be taken so that the need for (high) performance does not compromise any other requirements.

3.10 Reliability

The Toolkit must perform all functions reliably, without fail or errors regardless of the involved complexity. This means that its design must be robust and defend against input or system faults. Since security assurance is closely linked to trust this requirement is especially relevant to the Toolkit.

3.11 Scalability

Even though it is hard to anticipate actual usage-volume, the design can cope with increased demands and provides the capability for a future scale-up in terms of the supported features and number of users. This is achieved with built-in vertical scaling arrangements as well as a plan for more profound changes in the employed technologies using a proper modular design.

3.12 Security

All aspects of security are of particular interest and importance to the Toolkit. All users should be authenticated, communication should take place over encrypted channels, data must be safely stored and the system must be always available, within reason.

3.13 Testability

Testability ensures that parts of the Toolkit software can be tested in isolation or in combination. This can be done by adopting a coding style that allows for testable components, such as dependency injection [8], and by writing adequate tests (e.g., unit testing) to cover as much of the functionality as possible.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no 732319

4 Technology enablers and tools

4.1 Available solutions and guidelines

Numerous approaches could be taken to develop the AFT. What follows is a non-exhaustive list of possibilities along with a brief justification for their rejection.

- A native application is a binary which runs on a host computer. It was rejected due to its difficulty in updating and supporting collaboration between users.
- A mobile application is easier to update but constrains the users with regards to the device they can use and also requires extra effort due to the diversity between mobile platforms.
- A traditional web application may be easy to update and used for collaboration across different devices. However, server-side rendering [9] (i.e., the server's response to the browser call is the HTML description of the page which is ready to be rendered) suffers from some shortcomings like rigidity and higher hardware requirements.

4.2 Single Page Application

Our choice is to rely on a Single-Page Application[4] approach which has recently become very popular. Such applications offer a number of benefits which are very relevant to the Toolkit:

- They have distinct client and server components. The Toolkit then will have a clear separation between user interface and business logic thus ensuring modularity (Requirement 3.1 Adaptability, 3.6 Extensibility, 3.11 Scalability) and testability (Requirement 3.13 Testability).
- Due to their popularity, there are ample technologies and toolchains which are used to support the SPAs. Thus, the Toolkit implementation process will benefit from investing effort on the actual development (rather than set-up and/or configuration tasks).
- All communication between client and server code occurs with AJAX calls. As follows, the Toolkit will have a ready Machine-to-Machine interface (Requirement 3.1 Adaptability) ready without any extra effort.

A single-page application works by first delivering the web page along with all functionality in the initial browser request (see Figure 2). After that initial load the page updates itself leveraging small asynchronous requests to the web server (which improves performance, i.e. Requirement 3.9 Performance).

[4] https://en.wikipedia.org/wiki/Single-page_application

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no 732319

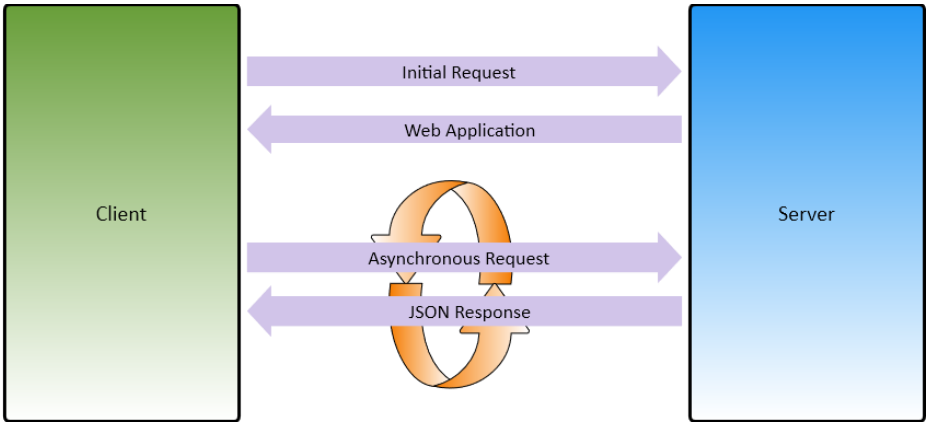


Figure 2: The SPA paradigm

The benefits of such an approach are multiple. This approach enforces design modularity as there is a clearly defined interface between client and server code. It also reduces server load as more functionality takes place on the client machine. Also, after a short wait during the initial load, the user experiences a significant performance improvement (Requirement 3.9 Performance). Finally, as with all web applications, SPA's can be used by all browser-equipped devices (Requirement 3.7 Interoperability).

4.3 Front end

In what follows the AFT front-end functionality along with relevant software modules are described.

4.3.1 The JavaScript ecosystem

The JavaScript programming language was created in order to enable interactive web pages. It has since grown-out of web browsers and now powers almost all kinds of applications. It enjoys wide deployment due to its ease of use and its relevance to modern web development. It has been standardised in ECMAScript [3] and implemented in various environments most notably browsers.

The Node.js run-time environment is a non-browser implementation. It is used for web hosting, server-side scripting as well as development. The node.js package manager or NPM [10] is a tool for managing dependencies in software projects and is responsible for the great wealth in JavaScript libraries which drives the modern web.

The Typescript language is a package created to improve modern applications. It is a superset of JavaScript and provides type annotations, type checking, classes, anonymous functions and many other features which are usually absent from scripting languages. It compiles to JavaScript and can run on most modern browsers and other environments.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no 732319

4.3.2 The Angular Application Platform

Angular is an application framework [12] which can be deployed in multiple systems and makes heavy use of Typescript. It has features such as data binding, templates, dependency injection and tooling in addition to all features provided by the language to facilitate development best practices. It thus leads to better applications with modularity (Requirement 3.1 Adaptability, 3.6 Extensibility, 3.11 Scalability), extendibility (Requirement 3.6 Extensibility) and testability (Requirement 3.13 Testability) already provided for.

An Angular application contains modules which in turn consist of components. Components contain application logic and templates which are used to create views as well as binding mark-up which link application data to Document Object Module (DOM) elements. Services also contain application logic shared between components by injection. One such service is the Angular router which constructs a navigation tree for the application in accordance to browser expectations.

4.4 Back end

In what follows the AFT back-end functionality along with relevant software modules are described.

4.4.1 .NET Core

Dot Net Core [11] is a multi-platform open-source software framework. It contains a compiler and a runtime environment as well as a package manager for developing and running C#, F# and VB applications. It supports console and web applications but not native user interfaces. The platform comes with built-in tooling to support the development process.

Specifically, for web applications the platform provides the ASP.NET Framework. It can be used both for web UI and web API modular applications with a focus on clean design and testability (Requirement 3.13 Testability). It interfaces well with all single-page application frameworks due to the Model-View-Controller component which is complete and well-implemented.

Another part of the .NET stack is the Entity Framework Core, an object-relational mapping library. Besides the built-in database providers there are others available for most popular databases. EF Core allows for both mappings on an existing database as well as table generation from existing models. The latter approach allows for easier integration into applications (Requirement 3.1 Adaptability) and also promotes testing (Requirement 3.13 Testability).

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no 732319

4.4.2 PostgreSQL

PostgreSQL is a well-known and widely used open-source, multi-platform ORDBMS focused on stability and correctness [17]. In addition to the usual database features it also supports many advanced ones. It is not supported in EF Core by default but there are stable packages which allow for an almost seamless integration. In all it provides for a robust storage option (Requirement 3.12 Security).

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no 732319

5 The proposed AFT architecture

5.1 Front-End

The front-end component is a composition of modules; the root module contains (numerous) feature modules resulting in a tree-like structure. The basic module architecture for the AFT front-end is shown in Figure 3. The module contains a template which describes the view and provides a two-ways binding to the component which in turn serves as the host for the application data and logic. Thanks to this binding any update in the underlying data is automatically reflected in the view and vice versa. Directives are instructions (i.e., a way to inject program logic) given to the templates to manipulate the view in specific ways. Services are reusable components which don't contain any view logic and can be injected to any components as required (e.g., import child components into where needed).

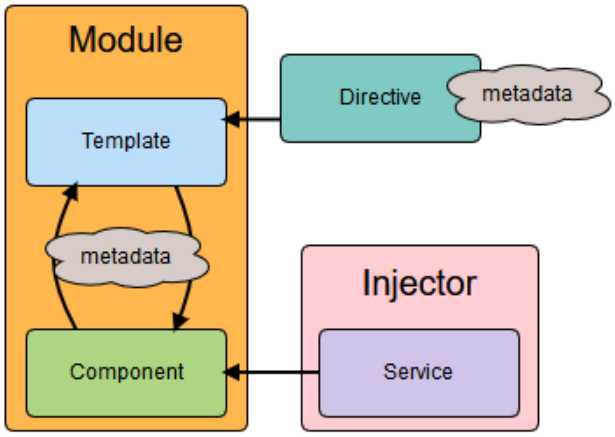


Figure 3: Basic Angular Concepts

The AFT front-end design adopts principles coming from the Angular platform for desktop web applications development, detailed in [12].

5.2 Back-End

The back-end will implement a Model-View-Controller (MVC) pattern. Communication with the client will be achieved by means of a RESTful interface. Requests arriving to the endpoint will be processed by the web framework's pipeline (see Figure 4). After the request arrives it is routed to the appropriate controller. A series of discrete steps produces the appropriate response to be sent back to the client. This process can be further customized by using filters between the different stages.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no 732319

Figure 4: The ASP.NET pipeline

The backend will also feature several distinct layers. The core functionality will be contained in the domain layer where it can be tested and verified independently (see Figure 5). This will lead to a solid and extendable implementation (Requirement 3.6 Extensibility). The object-relational-mapping will handle all saving and retrieving of information from the database and the domain layer will be free of database access logic.

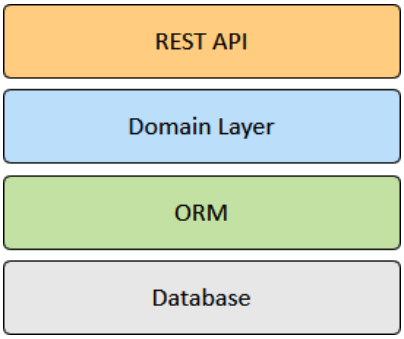


Figure 5: The AFT Stack

5.3 Action Example

This is an example of a partial implementation (see Figure 6). AuthController is responsible for authenticating the users and inherits a number of attributes from the Controller which is a framework class. The AuthController is then used to create the corresponding interface endpoint. The controller uses an interface to the user data and a cryptographic service (Requirement 3.12 Security) set up during the application start-up. Interfaces are very useful in this pattern because they provide inversion of control between the classes. CryptoService makes use of interfaces as well to save an encryption key which is used to sign all responses to the clients. Those interfaces are implemented by classes which are also instantiated during the start-up and then injected to the services. This way the service can be tested separately using mocks. The class PersistenceContext inherits attributes from the database framework and exposes the database in an object-oriented way.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no 732319

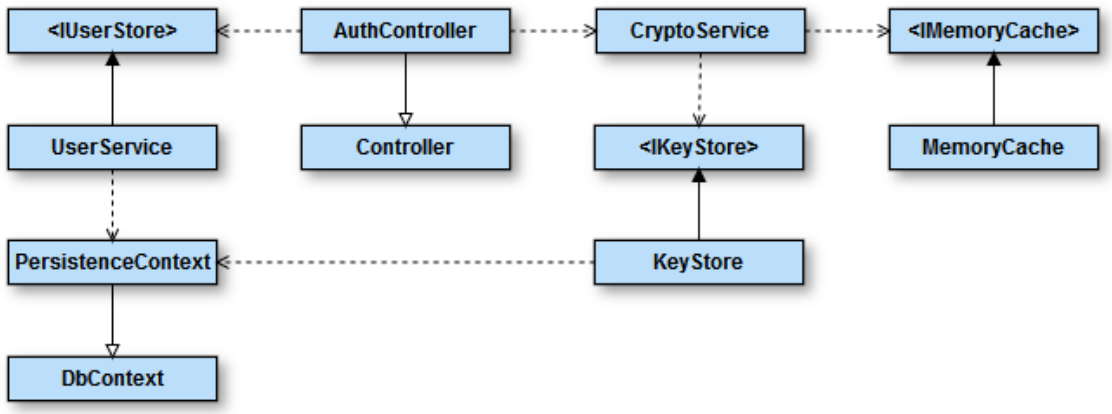


Figure 6: A Partial Example Implementation

5.4 Use Cases

In what follows a basic set use-cases that the toolkit (initially) supports are outlined. For each of the four identified use-cases the relevant functionality in terms of a series of processes in time (i.e., Preconditions, Flow and Postconditions) is explained. The Toolkit has been designed (and at this stage, partially developed) to efficiently support those use-cases but is not limited to them (see Appendix A - Potential Extensions).

1. Data Entry

Preconditions

1. The user is logged in
2. The user has the appropriate rights

Flow

1. The user selects the type of item to add
2. The user provides the item details

Postconditions

1. The item is added to the database

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no 732319

2. Data Edit

Preconditions

1. The user is logged in
2. The user has the appropriate rights

Flow

1. The user selects an item from the database
2. The user updates the item information

Postconditions

1. The new information is saved in the database

3. Security Target Definition

Preconditions

1. The user is logged in
2. The user has the appropriate rights

Flow

1. The user selects the product
2. The user defines the product
    1. The user provides a Target Of Evaluation
    2. The user provides Protection Profile conformities
    3. The user provides the standard conformity
3. The user provides the definition details
    1. The user provides the list of assets
        1. The user provides the list of threats for each asset
    2. The user provides the security objectives
    3. The user provides the Security Function Requirements
    4. The user provides the product functions

Postconditions

1. The Security Target is saved in the database

4. Product Specifications

Preconditions

1. The user is logged in
2. The user has the appropriate rights

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no 732319

Flow

1. The user provides the product interfaces
    1. The user links each interface to the Security Functional Requirements
    2. The user links each interface to the Product Function
2. The user provides the product modules
    1. The user links each module to the Security Functional Requirements
    2. The user links each module to the Product Function

Postconditions

1. The product specifications are saved in the database

5.5 Entities

To support its basic functionality reflected in the aforementioned use-cases 3 and 4, the Toolkit realizes a data modelling structure comprised by a set of entities and their relations. Those entities seek to accurately represent the building blocks (or sequence of required inputs) for the compilation of an ST in line with the SAFERtec Assurance framework (see [2] and the Deliverable D3.2).

The following diagram (Figure 7) represents the conceptual draft of the entities which will reside in the database and will be exposed for editing to the users. The core object is the Product around which all others will be centred. Most relationships are one-to-many.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no 732319

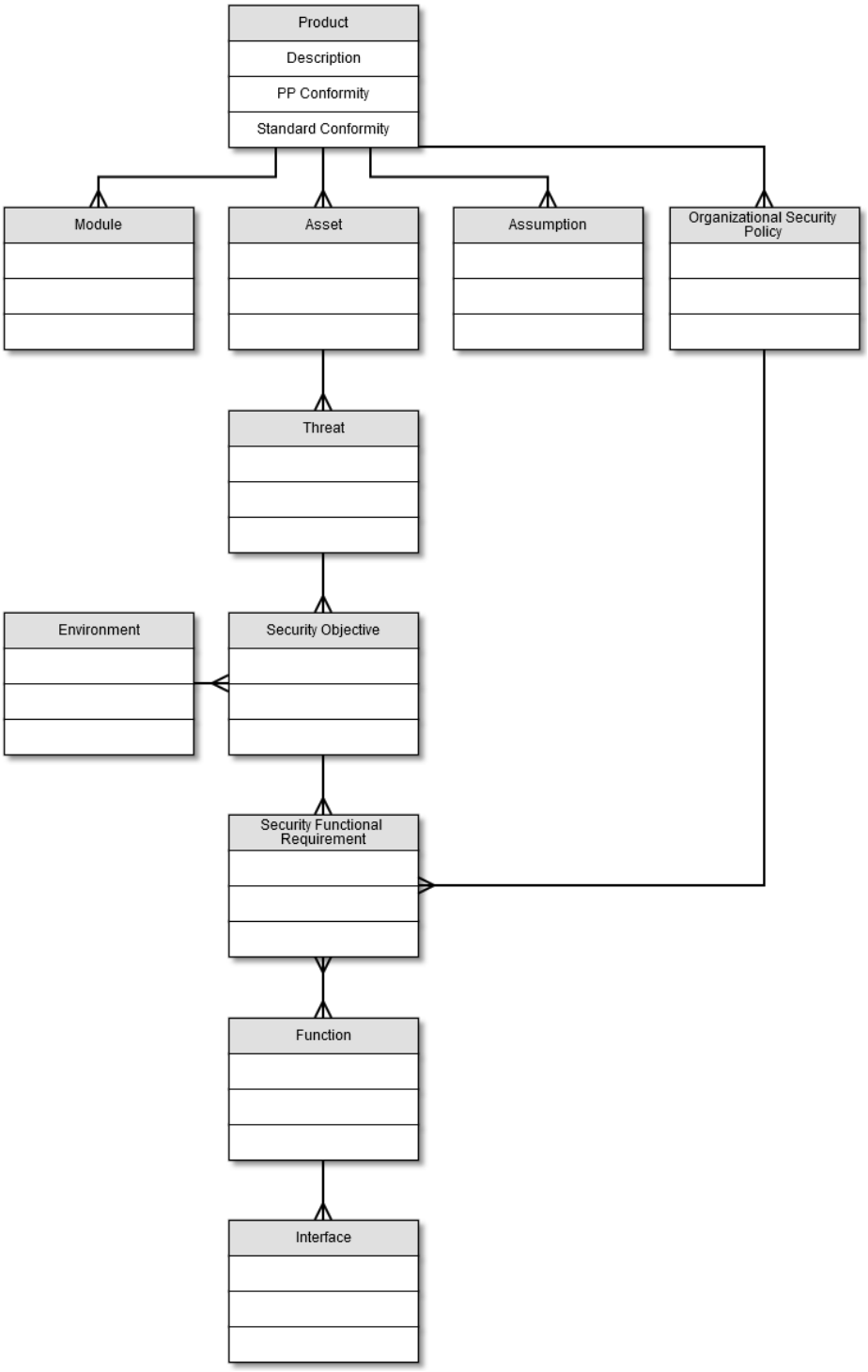


Figure 7: The main entities and their relations (E-R diagram) in the AFT database

With the above E-R modelling and relevant implementation the toolkit is enabled to assist the user in defining a Security Target for the interested 'Product' in line with the SAFERtec Assurance

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no 732319

Framework. However, further help may be provided to the AFT user extending the Toolkit' functionality in order to cover other Security Evaluation Tasks. Such functionality will be supported (Requirement 3.6 Extensibility) and partially implemented already by the SAFERtec project. At the same time, extending the toolkit further will serve as an excellent opportunity for SAFERtec exploitation activities.

6 Analysis of expected AFT features

In this section we provide some further insights on a set of expected AFT features that the proposed architecture is enabled to support. Relevant examples and explanatory figures are provided where applicable.

6.1 AFT Interoperability features

This is addressed in a dual manner. The SPA, which is the UI component of the toolkit, communicates with the logic component of the toolkit via a JSON REST interface (see the upper part of Figure 8). This interface can potentially be leveraged by other software instances (such as a mobile front end) which need to communicate with the toolkit (as shown in the figure's lower part).

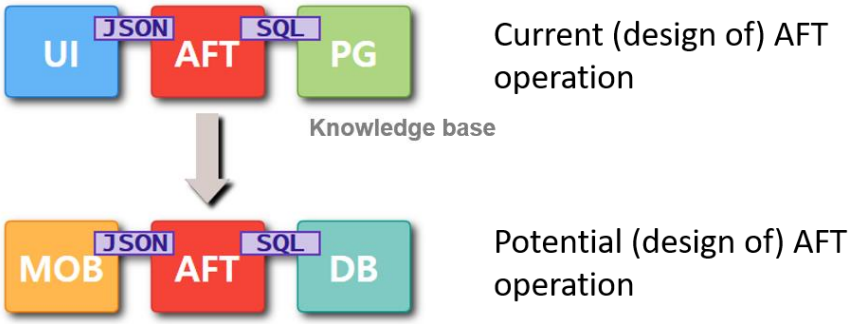


Figure 8 Showcasing the AFT interoperability

Secondly, the toolkit communicates with the database (i.e., in the upper part of Figure 8 AFT interacts with an SQL-compliant relational database management system) using standard SQL commands. Excluding some optimizations, which are minor in volume, the language used is generic which means that the specific database can be swapped with another one (see lower part) at the expense of minimal effort.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no 732319

6.2 AFT Modularity features

Modularity refers to the property of software to be composed of different parts which interface together through standard mechanisms. The AFT modularity is on the one hand reflected on a 'horizontal distinction' of its structure (see Figure 9) and on the other, on the usage of standard interfaces (as explained in the previous paragraph).

AFT is composed of a UI editor (further organized into distinct logical parts) which communicates with the main engine through a JSON interface. The main engine, which houses most of the program logic interfaces with the database through an SQL interface.

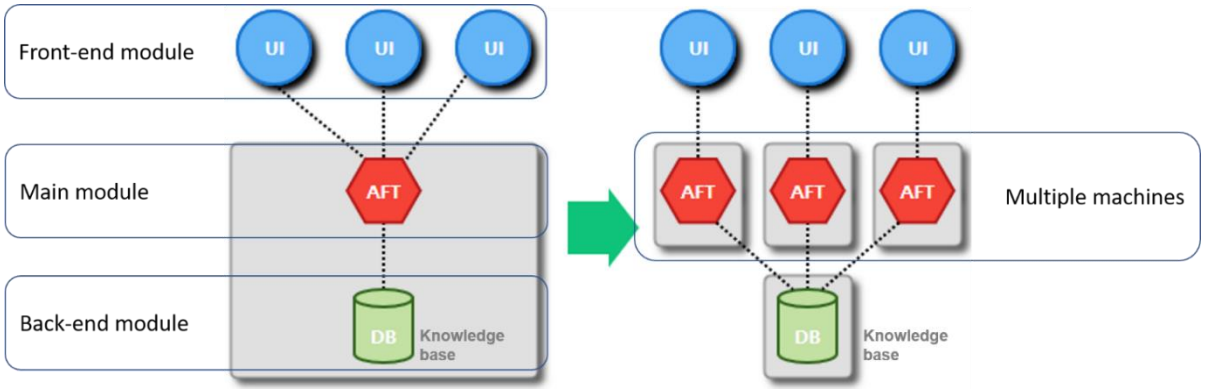


Figure 9 AFT modularity along horizontal axes eases its deployment across machines (on the right)

Such a design eases the AFT deployment over (physically remote) machines that use their own user interface and AFT main module while (potentially) accessing the same back-end database realizing the AFT knowledge base (see the right part of the figure).

6.3 AFT Extensibility features

The AFT toolkit provides a basic data frame for the efficient support of Common Criteria/SAFERtec evaluation of automotive products. In addition to those features (shown in Figure 10 as the AFT editors) provided in the toolkit now, more can be added to support extra evaluation classes (included either in Common Criteria or the SAFERtec assurance framework).

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no 732319

Figure 10 AFT can support any Common Criteria evaluation class, if extended

Future extensions will leverage the models and data provided now in order to extend the toolkit implementing functionalities that will facilitate other Common Criteria (or SAFERtec proposed e.g., AOP) evaluation classes. The available CC entities implemented in the toolkit can be leveraged by tools for other Common Criteria classes besides the Security Target.

Additionally, the toolkit can be extended with different user interfaces, such as mobile ones, by leveraging the REST API to connect to AFT.

6.4 AFT Computational Efficiency features

The nature of the AFT application allows for a variety of characteristics contributing to computational efficiency. The first is that the single page application offloads all computation pertaining to user interface rendering to the clients, alleviating the requirements for server resources. A single-page application delivers the web page along with all functionality in the initial browser request followed by small asynchronous requests to the web server to fetch (only) the necessary page updates; that's what improves performance compared to traditional server-side rendering applications [9].

Another point that relates indirectly to computations is the fact that most of the communication between the server and clients is lightweight resulting in smaller strain on network resources. Using the *XMLHttpRequests* object reduces communication overhead compared to the *PostBack* process; the latter originates from the client-side browser. The web page and its content are sent to the web server for processing and subsequently the web server 'posts the same page back' to the client [14].

Finally, the AFT knowledge base (i.e., a custom database) is tailored to the application, meaning that the transfer of information between the toolkit and the knowledge base is fast and efficient while the computational needs of the database are minimised.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no 732319

7 Conclusions

The software requirements and relevant architecture for the SAFERtec Assurance Framework Toolkit detailed in the present document, serves as a solid foundation for the development of a modular and extendable application. It has been designed to bear improvements made with minimum effort and enabled to build further features upon-it. The technologies that have been used are fairly well-established (in line with a number of relevant software standards) with broad support and advanced capabilities supporting rapid development and easy deployment.

Importantly, the design has been closely followed by the relevant skeleton implementation that is being continuously updated and checked against the SAFERtec framework development.

In this way, the introduced Toolkit can be efficiently implemented, tested and promoted for usage by the relevant audience. More ambitiously, thanks to its welcome design characteristics it can be easily maintained and extended even beyond the SAFERtec timeline.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no 732319

References

[1] ISO/IEC 15408 part 1/2/3:2005-Information technology, Security techniques, Evaluation criteria for IT security," Tech. Rep., v3.1, Release 5. [Online]. https://www.commoncriteriaportal.org/cc/

[2] P. Pantazopoulos et al., "Towards a Security Assurance Framework for Connected Vehicles," in *IEEE 19th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, Chania, 2018, pp. 01-06.

[3] ECMAScript Language Specification. [Online]. https://www.ecma-international.org/ecma-262/

[4] W3C Cascading Style Sheets. [Online]. https://www.w3.org/Style/CSS/Overview.en.html

[5] The JavaScript Object Notation (JSON) Data Interchange Format. [Online]. https://tools.ietf.org/html/rfc7159

[6] Scott Ambler. Technical (Non-Functional) Requirements: An Agile Introduction. [Online]. http://agilemodeling.com/artifacts/technicalRequirement.htm

[7] Data retention - European Commission. [Online]. https://ec.europa.eu/home-affairs/what-we-do/policies/police-cooperation/information-exchange/data-retention_en

[8] Yunwu, and John J. Ponzo, Huang, "Dependency injection by static code generation.," U.S. Patent No. 8,745,584. , june 3, 2014.

[9] Michael Alan, Kutner, "Server-side rendering," U.S. Patent No. 8,429,269. , April 23 , 2013.

[10] NPM. [Online]. https://www.npmjs.com/

[11] [Online]. https://dotnet.microsoft.com/download

[12] ANGULAR. [Online]. https://angular.io/guide/architecture

[13] Paul Reed, Jr., "Reference Architecture: The best of best practices" [Online] https://www.ibm.com/developerworks/rational/library/2774.html

[14] What is PostBack http://net-informations.com/faq/asp/ispostback.htm

[15] HTTP and reverse proxy server [Online] https://nginx.org/en/

[16] Kestrel Server [Online] https://stackify.com/what-is-kestrel-web-server/

[17] PostgreSQL Open source relational database [Online] https://www.postgresql.org/

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no 732319

Appendix A - Potential Extensions

The current implementation of the Toolkit involves the provision of support and means for an automotive developer to facilitate the efficient compilation of an ST (or accordingly a PP) for his/her product (ASE class corresponds to ST evaluation). The proposed architecture is extendable to include functionalities that may further assist the realization of other evaluation tasks in line with the classes introduced by CC:

- ADV (Specification). The aim is to help the developer provide the functional description needed by the security evaluation process to relate the identified SFRs to the ToE's interfaces (TSFI), and then to the included modules.
- ATE (Functional tests). The aim is to help the developer justify how every test he/she is performing correctly corresponds to checks for the TSFI and SFRs

Regarding ADV the Toolkit can be extended to include the different elements that compose the functional architecture of the considered product (ToE). Typically, this corresponds to the breakdown of the involved code (e.g., a classes-structure in case of object-oriented implementations and a clear description of the various functionalities implemented in a configuration module, HMI module, computation module, etc.). The developer needs to be able to document the expected behaviour at a high level by describing the different input (data) types, the associated output and the potential error messages. Some further details should be provided in terms of modules and the way they communicate (i.e., interfaces) where justifications on the extent to which (e.g., partially, fully or on a supporting level) they implement the identified SFRs, are needed.

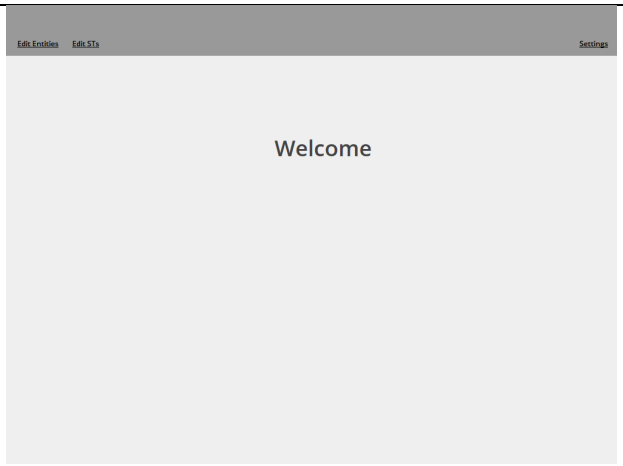
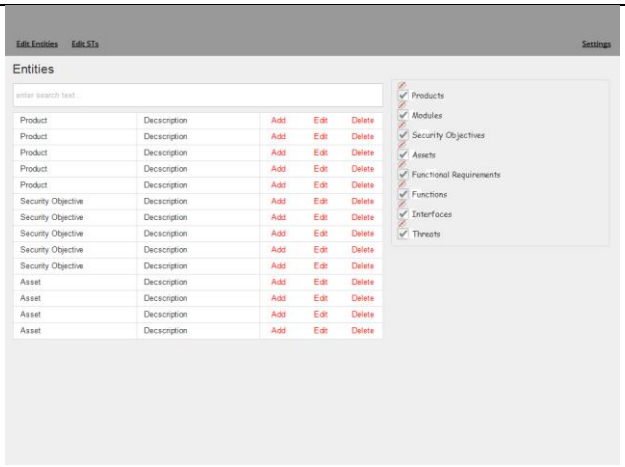
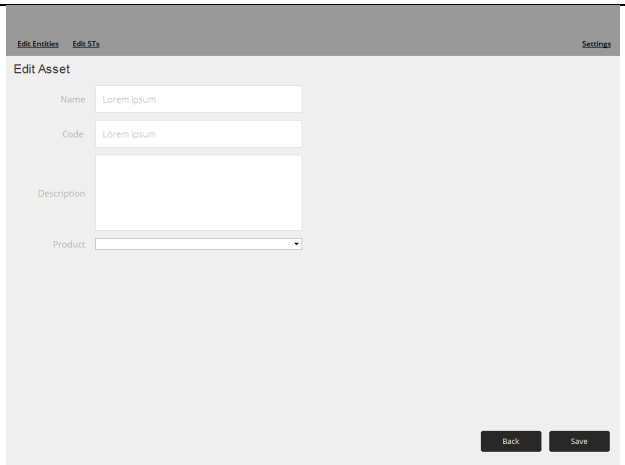
Implementation-wise, a representation of the ToE's modules is needed together with their interconnection through interfaces to allow the (ToE) developer describe the functional architecture of the product and specify whether and how the identified SFRs relate to those interfaces.

Regarding ATE, what is required for the developer is to describe the testing environment; required in the ST in the TOE description part (that could be an independent entity in the model). In the ST what is required is to describe every software and hardware required by the TOE to work, which actually describe its environment to which the environment used for the tests in ATE should be conformant. Then, for each test the developer has to describe, its goal, the initial set up including dependencies with other tests (e.g. a signature verification requires first to run the tests that generated the signature, etc.), the different steps, the expected results (e.g., files generated, messages excepted, packet sent, etc.) and finally to specify which TSFI and module the certain test relates-to.

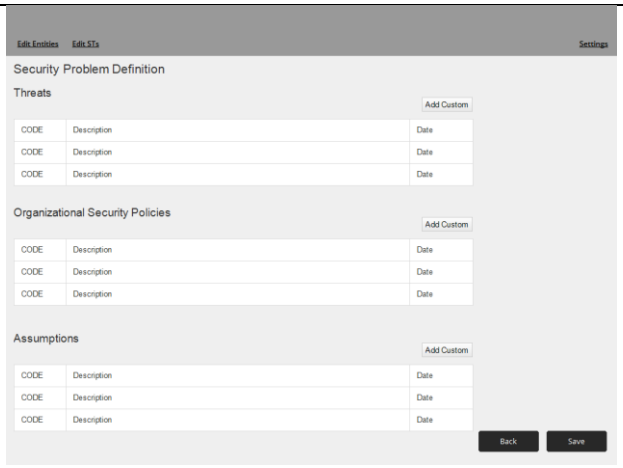
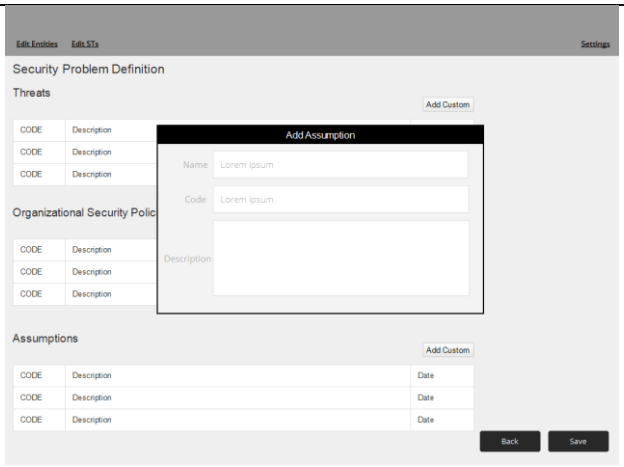
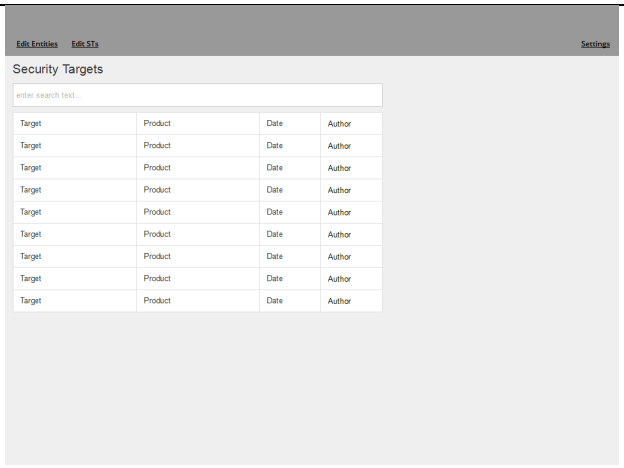
Implementation-wise, the ATE class can be covered by adding the test entity (and all needed associated entities) in the toolkit architecture to accurately describe the aforementioned testing functionality. Test entities should also be linked to interfaces and modules.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no 732319

Appendix B - Snapshots of the Toolkit Skeleton

Welcome screen	
Entities	
Edit an Entity	

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no 732319

Security Problem Definition	
Addition of an Assumption	
Security Targets in the Toolkit	

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement no 732319