

A12

Low Code for Individual Enterprise Software

2025.06 LTS

Legal notice

Authors:

Hamarz Mehmanesh
Sabine Gandenberger
Ansgar Weiss
Thomas Kneist
Sebastian Lorenz
Martin Backschat

mgm technology partners GmbH

Taunusstr. 23
80807 Munich
Germany
Tel: +49 (0)89 / 358 680-0

Jurisdiction and place of performance: Munich

All rights reserved.

Permission required for any reproduction, even excerpts.

© 2025 mgm technology partners GmbH

www.mgm-tp.com

Contents

01	1. Executive Summary	4	04	4. Modeling Platform	22
	1.1 What is A12?	5		4.1 Modeling: A New Software Development Discipline?	23
	1.2 For Which Operational Scenarios Is A12 Designed?	6		4.2 The Modeling Concept of A12	23
	1.3 What Benefits Does A12 Provide?	7		4.3 Advantages of “Data First” Modeling with the A12 Rule Language	31
				4.4 Simple Model Editor (SME)	33
				4.5 Provision of the Modeling Environment	36
02	2. Motivation and Approach	8			
	2.1 From Micro Apps to Integrated Enterprise Applications	9			
	2.2 The True Cost of Enterprise Software Development	9			
	2.3 Model-Driven Development	14	05	5. Runtime Platform	37
	2.4 Digital and Data Sovereignty	15		5.1 A Different Range of Tasks for Developers	38
	2.3 Artificial Intelligence as a Catalyst	16		5.2 Architecture	40
				5.3 Components	34
				5.4 Project Template	56
				5.5 Operations	57
03	3. Plasma UI/UX Design System	18			
	3.1 Efficient User Interfaces for Enterprise Software	19			
	3.2 Methodology for Coherent User Experience	19	06	6. Appendix A: Technologies	60
	3.3 Accessible Web Applications	19			
	3.4 Reusable Widgets	20			
	3.5 Well-designed Theming Concept	20			

01

Executive Summary

A12 is a platform for developing enterprise applications in complex IT landscapes. It relies on model-driven software engineering (MDSE) and brings the low-code principle to the world of enterprise software.

This white paper presents the A12 approach and demonstrates how it can be used to turn applications into fully integrated enterprise applications that will work for a long time to come.

1. Executive Summary

Structure of the A12 Platform



A12 Modeling Platform

Tools to design domain contexts and parts of the user interface (Low Code)



A12 Plasma

Design System, Widget Library & Theming, specifically tailored to Enterprise Applications



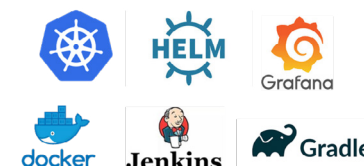
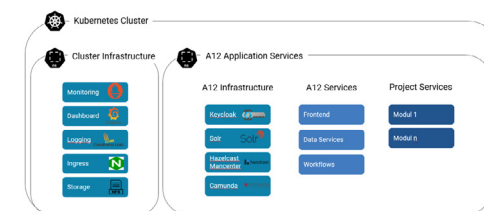
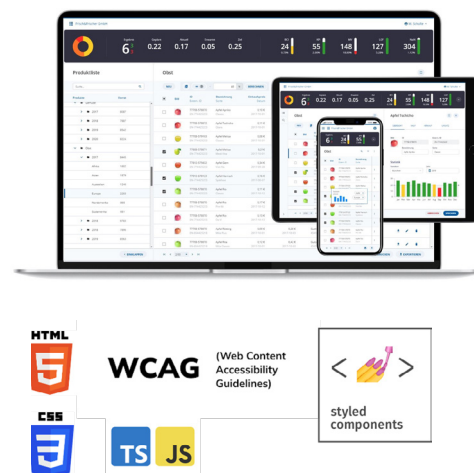
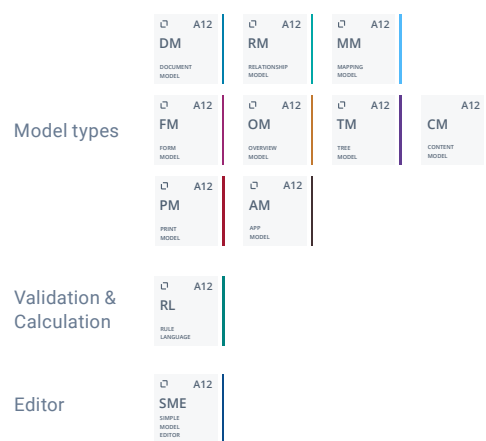
A12 Runtime Platform

Client- and server-side technical components for robust and scalable applications



Containerization

DevOps tooling for CI/CD and flexible on-premise and cloud operations



1. Executive Summary

For Which Use Case Scenarios Is A12 Designed?

Development of complex custom software

A12 is the ideal foundation for highly scalable, secure, robust, and potentially business-critical web applications—whether eGovernment application systems, portals, and specialized procedures for the public sector, underwriting platforms for industrial insurance, law firm management systems, or e-commerce applications.

Replacement of Excel/Access applications

Despite all the disadvantages, many business processes are based on Excel and Access solutions. With A12, they can be consolidated and transferred to an open platform as real applications (including automated migration tools). This simplifies management and further development and ensures compliance with regulations.

Core-level platform for an integrated IT landscape

Medium-sized and large companies as well as public authorities can benefit greatly from consolidating and modernizing their IT landscape on the basis of a uniform platform. Thanks to its modular structure, robust basic components, and focus on flexible system integration, A12 is ideally suited for this purpose.

Software product development

A12 provides a proven foundation for developing long-lasting software products. Using A12 shortens time to market and simplifies the development of a high-quality application that is not only state-of-the-art for a brief moment, but remains so thanks to the continuous development of the platform infrastructure.

1. Executive Summary

What Benefits Does A12 Provide?



Handling your business content yourself

Business experts and analysts can use the modeling tools themselves to create the software's domain-specific core and maintain it in the long term.

- Adjust business aspects without programming knowledge
- Implement business changes rapidly
- Automate the software development process extensively
- Powerful Rule Language for validations and computations



An open platform, not a closed ecosystem

A12 is designed as an open system. It provides an enormous amount of flexibility for software development, long-term maintenance and further development.

- Flexible use of modular runtime components
- Systematic use of open source technology
- APIs for individual extensions at any level
- Full operational control – on-premises or (private) cloud-based
- Controllable and responsible use of AI



Future-proof platform for long-lasting software

The consistent separation of business-specific models and technology makes it possible to retain the business-specific core even in the case of technological leaps.

- Detached innovation of technology through model-based approach
- "Data First" principle for sustainable domain-oriented modeling
- Careful technology selection and use of industry standards
- Continuous development of the technical basis

02

Motivation and Approach

mgm's goal is to faster and more economically build enterprise software that is robust, secure and durable. Our experienced software engineers back up this claim. They work at gradually reducing the typical expenditure involved in developing enterprise applications. This is first and foremost done by using model-driven abstractions and separating business expertise and technology.

2. Motivation and Approach

From Micro Apps to Integrated Enterprise Applications

Many enterprise applications generally originate as pragmatic makeshift solutions in separate business departments. More often than not, they start off as small Excel tables. They get bigger and bigger, incorporate macros and end up becoming almost like applications themselves! This pragmatic approach ("shadow IT") has its downsides, namely major data protection and IT security risks.

Low code platforms aim to eliminate the potential breaches caused by makeshift solutions with applications. They give business departments the opportunity to build their

own real applications, taking into account company-specific IT guidelines. This step is perfect for those makeshift solutions that have particular potential. Another challenge arises for another subset of micro apps, of course, usually for the ones that are most critical to business: they must be integrated into a heterogeneous IT landscape. Most low code platforms come with turnkey solutions for the most common integration scenarios. But they do have their limits. Custom development and professional system integration are unavoidable.

A12 not only helps you with the transition to micro applications, but also with moving towards integrated enterprise applications. In the long term, this is where companies spend the most money: in the development, maintenance and operation of enterprise applications.

The True Cost of Enterprise Software Development

mgm has been developing custom enterprise software for over 25 years. The core concept behind A12 is based on a series of observations that we made over and over on a wide variety of projects. Most importantly, the usual cost drivers (business adjustments and integrations) that occur once a project has been started are constantly underestimated and can end up being the reason why IT projects fail over the long term, even very large ones.

Enterprise domains – anything but standard

Every enterprise software program models a certain aspect of a company's reality. The particular model is based on the enterprise domain. The enterprise domain comprises a set of (enterprise) entities. These might be things like clients, products or orders. Each of these entities is represented by an entity model in the software. This model defines the entity's structure, attributes and relationships with other entities. Entity models are subject to constant change, which is often a major cost driver in enterprise software development.

The following points are primarily responsible for change:

- ⊕ A growing business is a complex network. All the company knowledge is spread out in different people's brains. There are many factors that influence the business that individual company representatives deal with over and over again.
- ⊕ The company continues growing. The portfolio changes. New distribution channels are added; others are eliminated. Different branches have to adhere to new regulatory requirements.
- ⊕ Each company organises their business in their own way depending on a variety of different rules. What's more, they all use their own terminology, which is constantly growing and changing.

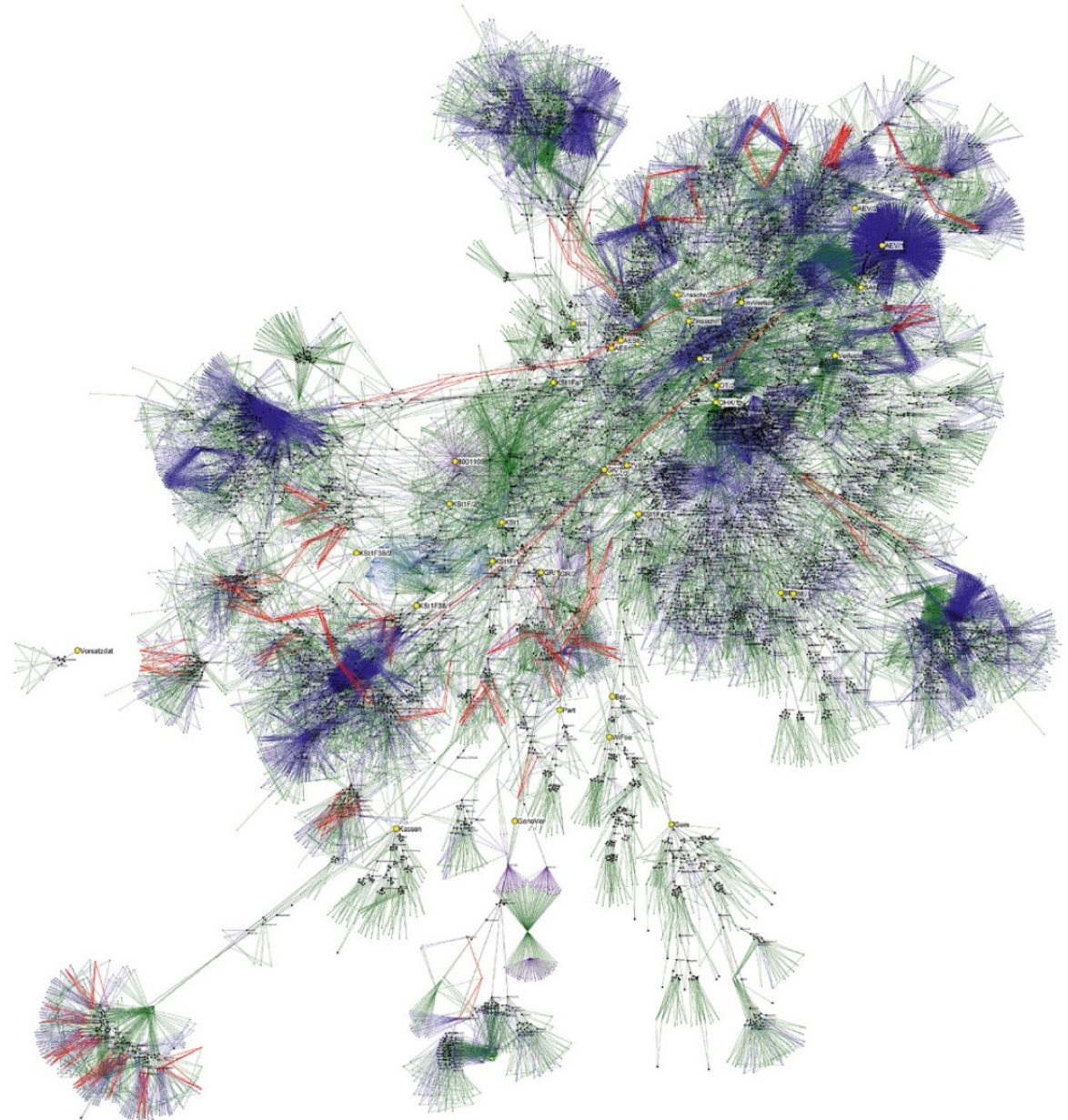


2. Motivation and Approach

And so, the models on which different enterprise software programs are based do not follow the same standards. On the contrary, they are highly individualised, always have exceptions and sometimes even inconsistencies.

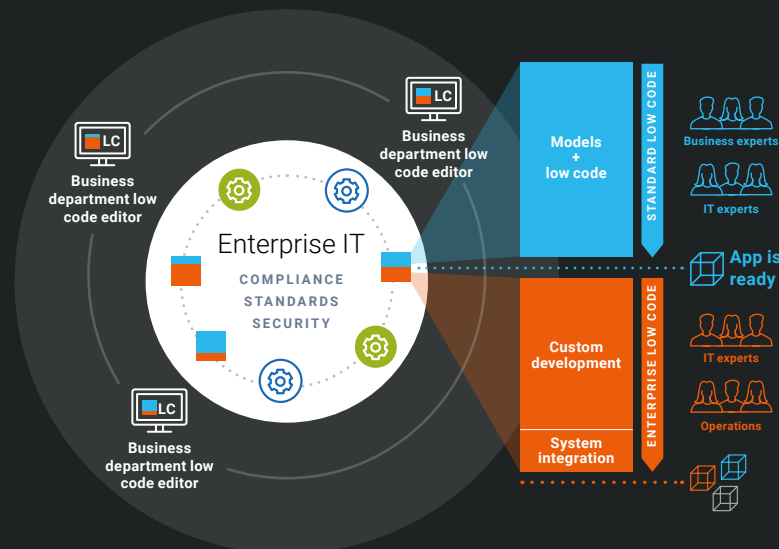
Models are yet another cost driver. Models gradually get more complex. They map aspects of reality that are constantly growing and that are relevant to the respective enterprise domain's success.

The figure illustrates just how complex they can get, for example when mapping a tax form.



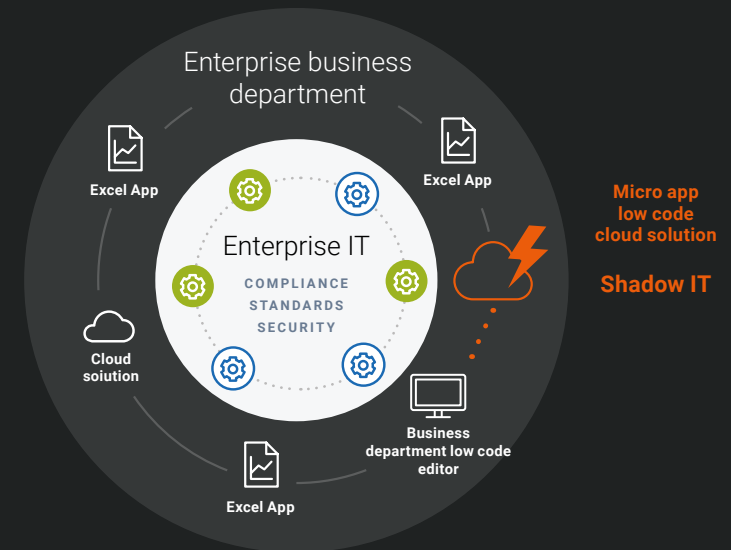
Enterprise IT and Shadow IT

Classic Enterprise IT in companies is structured centrally. The standard software and all custom-developed software used must comply with certain standards. But people are finding more and more solutions outside enterprise IT for department-specific requirements. They are created as Excel-based solutions or basic micro apps, for example. This often happens without telling the IT department. And thus shadow IT is created.



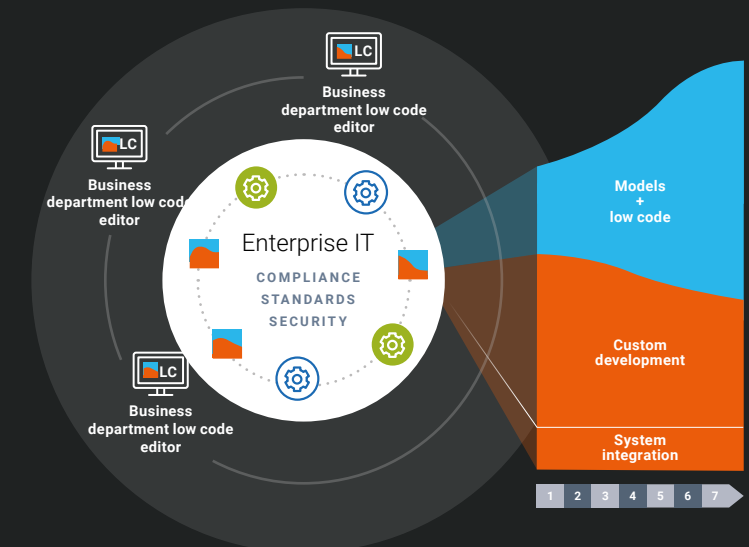
Dynamic weighting of low code and custom development

Custom Enterprise Projects require project-specific and customised methods. There can be a lot of variation in the weighting of low code and custom development from project to project. But the weighting can also vary considerably throughout the different life cycle phases of an IT project.



Enterprise low code development

If departments can design their own applications with the right low code tools and the IT department can secure the technology and standards centrally at the same time, enterprise applications with true value can be created. The low code part can be individually weighted depending on the project. The goal is to get the department and IT experts working together effectively.

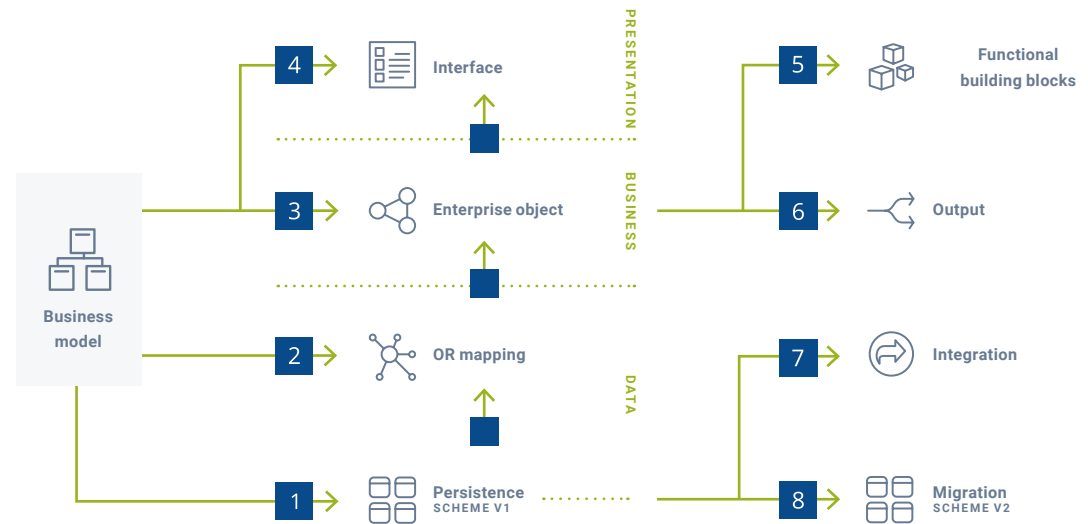


2. Motivation and Approach

Different Representations of Enterprise Entities

Any change in an enterprise entity leads to more expenditure from a software development point of view. Why? Because they have to be represented differently in different technical contexts. And you need mappings between these different representations. In practice, one tiny business change can create a domino effect of adjustments that need to be made to the software.

The figure to the right gives an overview of the various representations and mappings that are generally found in enterprise software.



The **business model** is the starting point. It is a model for implementation – the first representation of the modelled enterprise entities and their relationships to each other.

The following representations occur in a three-tier architecture:

- The enterprise objects are stored in a database in the **data tier**. Therefore, the representation needs to meet the requirements of this persistence level (1). The data is presented in tabular form in a relational database. Object-relational mapping is required so that the table data can be processed in an object-oriented high-level language like Java (2).

- The application logic is stored in the business tier. It has its own representation (3), which results from the respective processing of the enterprise objects, components and workflows.

- The **presentation tier** requires yet another representation (4). This deals with how the enterprise entities are presented in the user interface and how you can interact with them.

In addition, further representations and mappings of the enterprise entities are necessary in the following contexts:

- **Providing services for specific functionalities** – for example, checking stock (5).
- **Generating Word or PDF documents** such as insurance policies and administrative notices (6).
- **Integrating the application into other systems** in the company's IT infrastructure (7).
- **Extensive migrations**, that become necessary due to further developments of the schema of the underlying database (8).

2. Motivation and Approach



Each of these representations comes with its own set of challenges and costs. Some of them only become apparent after a while.

For example, the figure shows the fact that enterprise applications do not usually stand alone. And when they do, it's usually not for long.

On the contrary, they are usually integrated into complex IT landscapes and only realise their full potential when they are linked to a range of internal and external applications.



Heterogeneous IT Landscapes

The IT landscape of medium-sized to large companies across all industries all have one thing in common. They comprise large applications such as SAP or larger custom applications with a variety of smaller ones. In a perfect world, all applications would be fully integrated with regard to their involvement in processing business transactions and data exchange. But, because the technologies used in applications are so different (SAP, Java applications, cloud-based applications, etc.), these IT landscapes are usually built on a **technological basis that remains heterogeneous**. More specifically, this means that:

- Individual applications in this landscape usually have their own project team, release cycles and technology bases.
- Different technology and architecture decisions are made for custom applications depending on the application's age and the project team's preferences and decisions. This also applies to integrated applications that are based on applications such as SAP or MS Dynamics.
- Different applications usually also have different contact persons on the business side of things. These people draft the business specifications for the application's initial and subsequent development.

The heterogeneity of the IT landscape is another cost driver that even today's low code approaches cannot completely resolve. Custom development work can be reduced, but not completely done away with. Even if enterprise applications start off small, integration issues usually arise sooner rather than later as applications can only reach their full potential if they are integrated.

2. Motivation and Approach

Model-Driven Development

The many different representations of enterprise entities in the various software tiers are major cost drivers in enterprise software development. How can the cost of mapping these representations be reduced? Model-driven software development provides an answer to this question. The idea behind it is modelling enterprise entities and their relationships to each other. These models can be defined and adjusted using specialist editing tools.

Special interpreters and code generators translate the models into the application.

The kicker is that the elaborate mapping of different representations no longer needs to be done by hand. The generators and interpreters do it. This means that business content, which is subject to constant changes in enterprise software, as previously mentioned, can be displayed in the software much more quickly and with less overhead.

Advantages of Model-Based Development



✓ On-schedule implementation

Model-based development makes it possible for IT systems to be implemented and delivered on time, even when business requirements change frequently.



✓ Simplified dependency management

Model-based overall architecture simplifies managing dependencies. This makes it possible to separate business expertise and technical framework into separate release cycles. Furthermore, business expertise can be broken down into specific models for each version and data type. Each of these models is explicitly versioned, but is not dependent on version and data type. Business changes can also go live independently of each other for each data type and version.



✓ Less testing overhead

Testing can be extremely costly for custom-developed software that is constantly being changed. Each version, data type and change must be tested separately for each product. However, model-based development reduces the need for business testing, which is limited to the models.



✓ Clear path for technical innovation

As business expertise and technology are separate, technical innovations can be made without having to consider all of the application's technical content. For example, you can roll out new technology in the user interface design and implementation, in persistence or in server processing.

2. Motivation and Approach

Digital and Data Sovereignty

As a long term partner of public administrations, we support our customers in their desire for a self-sufficient approach to software. Business departments can do this with A12; they can keep full control, even when the applications are highly complex and integrated.

Control over business expertise - technology interchangeability

The strict division between business expertise and technology provides great flexibility for further software development over the long term. The business models make up the software's core; they can also be adjusted and developed without the help of a software developer. They are available as simple, open format JSON files.

This separation of the business content makes it much easier to change the technology. This would not be the case if the business aspects were closely interwoven into the code. One particular advantage is that the code does not need to be completely rewritten with each technical innovation. It is much easier to keep the technology up to date.

Control over data

For business-critical software, it is essential that sensitive data is stored in a trustworthy, secure environment and that smooth operation is guaranteed. We know just how important it is to have control over your business; we see this over and over again with our customers in the e-commerce sector. During periods of higher demand, like around Christmas, the systems run at maximum load for a long time without any downtime. Which is why, on the one hand, software must be scalable and high performance, while on the other, sole control over the underlying infrastructure and the release stages involved is also necessary.

We offer the following options for A12 installation:

- ✓ **On-premises operation in the company's own data centre**
- ✓ **Operation on mgm's private cloud, hosted in a German data centre**
- ✓ **Cloud operation with any cloud provider**

2. Motivation and Approach

Artificial Intelligence as a Catalyst

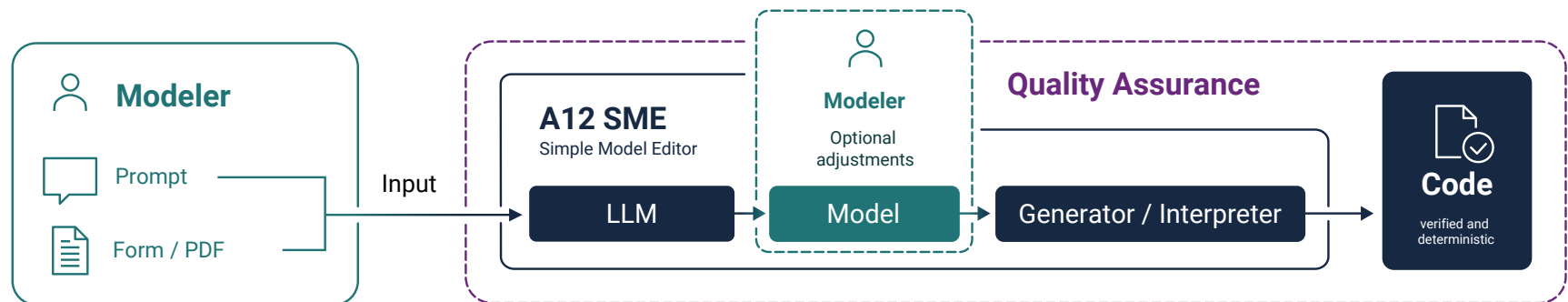
AI-based development approaches such as Vibe Coding are revolutionizing the software development process. However, due to the existing weaknesses of LLMs, such as hallucinations, they are (still) risky and can only be used in an enterprise context under strict conditions. Based on A12, however, many AI capabilities can already be tapped for enterprise development today.

In addition to robust components and clearly structured A12 models, mgm has developed a comprehensive quality assurance landscape for A12 as well as tools for security testing and build & deployment. This provides a controllable framework in which generative AI and agent-based systems can be used very effectively and responsibly.

Use of AI at the model level

A12's model-based development approach, combined with AI, offers a number of key advantages over conventional development paradigms. For example, AI can be used to generate models automatically. This reduces manual effort and speeds up the implementation of business requirements. For A12 Document Models, which are at the heart of data modeling, this capability has already been implemented as a prototype. In the Simple Model Editor, modelers can specify a source document and a prompt, which are used to automatically create an initial model that can be refined manually if necessary.

The particular advantage of using AI at the model level is that the regular processing chain of models in the form of interpreters and generators continues to be used. The resulting code remains deterministic. In addition, a range of existing test and quality assurance procedures can also be applied to generated models – e.g., the consistency check for validation and computation rules integrated in the Simple Model Editor (SME).



2. Motivation and Approach

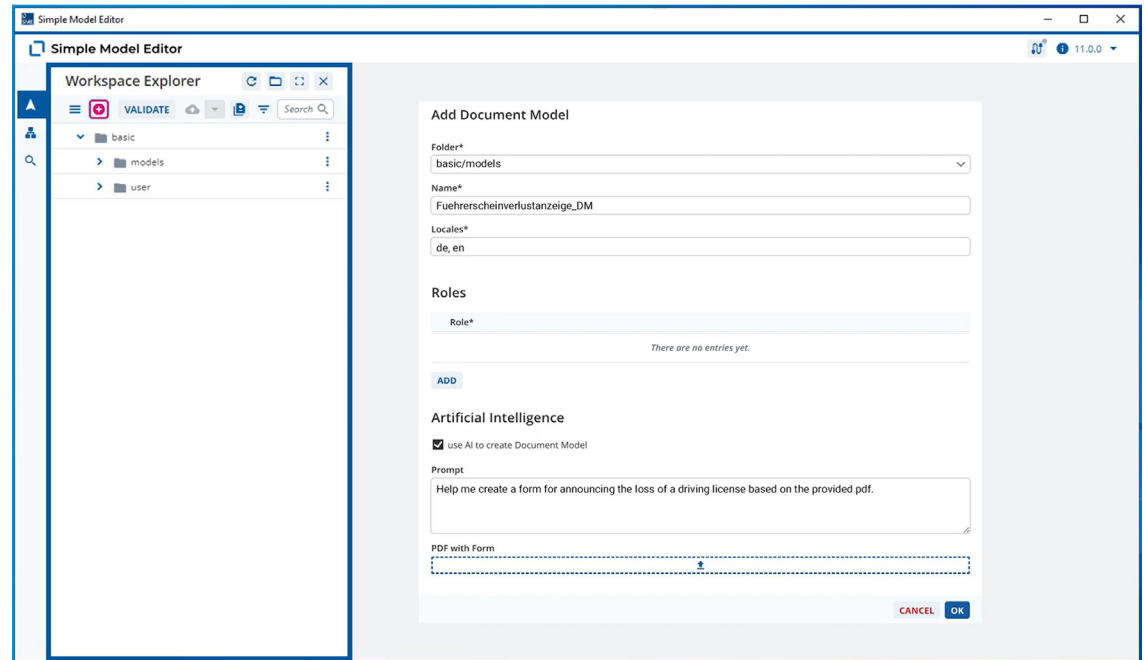
Use of AI at the code level

The use of AI at the code level also offers advantages for A12 projects. The components of the runtime platform provide stable and continuously tested solutions for many typical challenges faced by business applications – from authentication and authorization to accessible widgets. Using these components reduces the risk of security vulnerabilities and increases the productivity of AI bots. Extensive documentation of the components and specifications such as development guidelines and security guidelines help to ensure compliance with best practices.

Nevertheless, additional security measures are essential when using AI at the code level in A12 projects. These include, above all, code reviews. In general, no line of AI-generated code should be included in the project without human review. However, the test and security tools specially designed for A12 provide an additional safety net and help minimize the risks of AI in software development.

Measuring and controlling LLM results

In AI use cases, the question always arises as to which model offers the best results and how good the LLM results are. mgm has developed a series of procedures and schemes to continuously measure and evaluate the results of LLMs. These are an important building block in A12 projects for implementing AI-driven functionalities with clearly defined quality gates.



The screenshot shows the 'Simple Model Editor' application window. On the left is a 'Workspace Explorer' panel with a tree view containing 'basic', 'models', and 'user' folders. The main area displays the 'Add Document Model' dialog. This dialog includes fields for 'Folder*' (set to 'basic/models'), 'Name*' (set to 'Fuehrerscheinverlustanzeige_DM'), and 'Locales*' (set to 'de,en'). Below these is a 'Roles' section with a 'Role*' field and a message 'There are no entries yet.' An 'ADD' button is present. The 'Artificial Intelligence' section has a checked checkbox 'use AI to create Document Model', a 'Prompt' field with the text 'Help me create a form for announcing the loss of a driving license based on the provided pdf.', and a 'PDF with Form' field. At the bottom right are 'CANCEL' and 'OK' buttons.

03

Plasma UI/UX Design System

Enterprise applications are characterised by high information density and great complexity. Design languages such as Material Design hit their limits quickly. They cannot fully respond to some challenges, such as how to present complex tables, clearly. Or how to develop the user interface structure consistently when new information is added. This is why mgm developed A12's Plasma design system.

3. Plasma UI/UX design system

Efficient User Interfaces for Enterprise Software

Plasma comprises a variety of UI/UX components, usage patterns and design guidelines that can be used to design consistent, efficient and attractive user interfaces. And thus Plasma provides solutions for two of the main UI requirements in enterprise applications: scalability and complexity.

One of the main ideas behind Plasma is to reduce the represented information density as much as possible. Ideally, users are only presented with what they really need for the tasks at hand. They can work faster and more efficiently.

Methodology for Coherent User Experience

Plasma also has a variety of reusable models and components for requirements that appear repeatedly in enterprise application user interfaces – from log-in screens to validating user input. This includes models for the application framework, navigation elements and notifications, as well as concepts for handling enterprise objects and the standard workflows in which they are integrated.

Accessible Web Applications

The importance of accessibility in web applications is increasing. For several years now, public authorities within the EU have been obliged to make websites and mobile applications accessible. From 2025, according to the European Accessibility Act, with a few exceptions all websites and web applications must be accessible. With Plasma, the A12 platform is explicitly designed for building accessible web applications. Numerous UI components - including the model-driven engines for forms and overviews - are accessible out-of-the-box. However, in the project practice of individual software development, there are always additional aspects to consider. There are specific requirements that a Low Code platform per se cannot cover. For this purpose, the A12 team offers projects practical assistance in the form of a regularly updated guide. It contains, for example, background information on accessibility certification, design specifications, and requirements for modeling and development.

3. Plasma UI/UX design system

Reusable Widgets

Widgets are reusable components that follow Plasma design conventions and UX concepts. They support business applications running on desktops, tablets, and smartphones with keyboard, mouse, and touch input. The components offer an easy-to-use, well-documented, strongly typed API and are extensible and customizable.

Features:

- ⊕ Enterprise UI for web applications
- ⊕ Support for desktop, tablet, and smartphone
- ⊕ Keyboard, mouse, and touch
- ⊕ Accessibility
- ⊕ Compatible with all major browsers
- ⊕ Strongly typed programming API in TypeScript
- ⊕ Design extensibility: From the application level down to every single detail per component
- ⊕ Seamless integration with A12 engines and services

Well-designed Theming Concept

Plasma relies on an easily extensible theming concept that allows the look and feel of an application to be customized in no time. Thanks to the technical implementation with Styled Components and React, there is no need to load CSS files. A theme can be activated ad hoc by setting a JS variable. Plasma includes four themes out of the box, each supporting a dark and light mode:

`defaultTheme` — classic

`flatTheme` — modern

`compactTheme` — compact

`flatCompactTheme` — modern & compact

Themes not only encapsulate style guidelines, they can also be combined with each other. The existing basic themes are the ideal starting point for covering project-specific requirements.

Theme Switching

Create theme:
Use of `createTheme()` with individual configurations

```

1 import { createTheme } from "@com.mgmt.a12.widgets/widgets-core/l
2
3 const theme = createTheme({
4   colors: { primaryColor: "red" },
5   baseTheme: "flat-compact"
6 });

```

Expand and customize:
 Typography: `font`, `fontSize`, `fontWeight`
 Spacing: `spacing`, `horizontalSpacing`, `verticalSpacing`

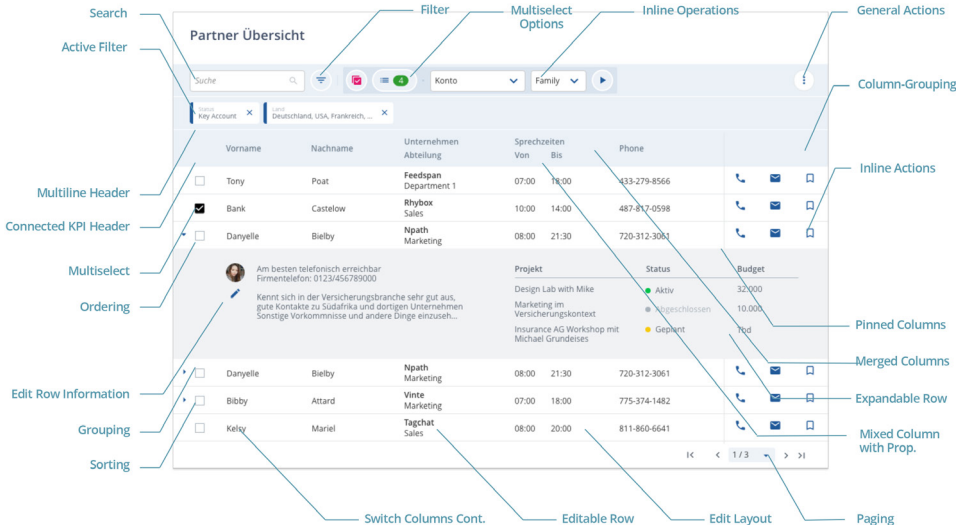
```

components: {
  button: { primary: { background: "orange" } }
}

```

Example: Component styling: change buttons specifically, for example

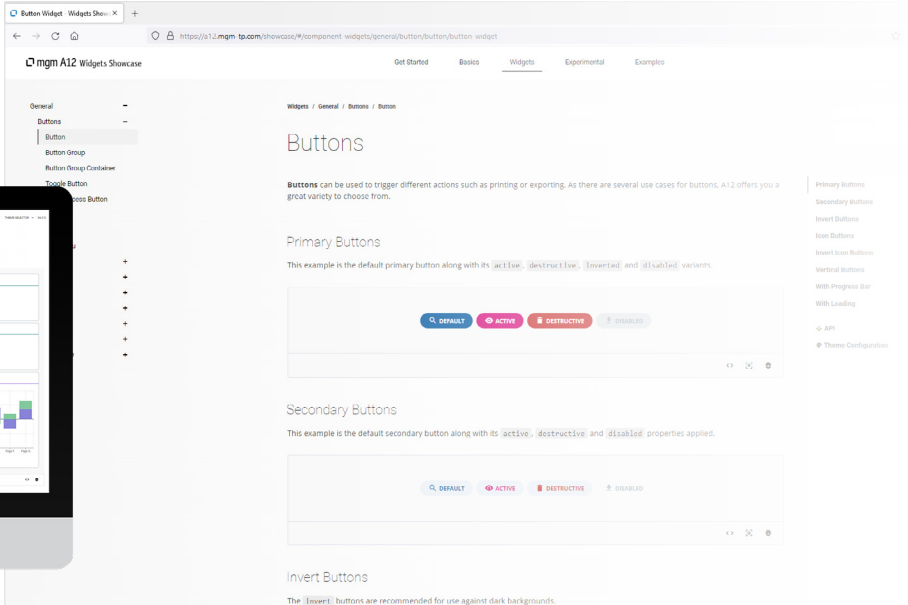
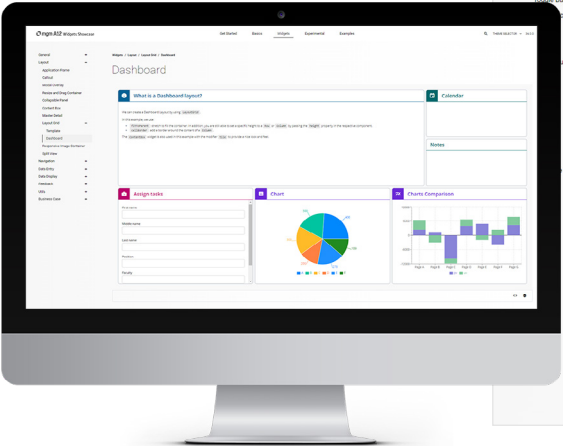
3. Plasma UI/UX design system



Unlike pure design languages like Material Design, Plasma also includes an extended range of functions which enterprise applications usually require. The figure provides an example of the overall concept for tables and all common features. We have already implemented some of these features in Plasma; others are still in progress.

A12 Widget Showcase

<https://a12.mgm-tp.com/showcase/#/>



04

Modeling Platform

The A12 modeling platform provides several modeling tools and a rule language that can be used to map high levels of business complexity for enterprise applications. The following sections provide a brief introduction to the modeling philosophy of A12 and introduce the main models and tools.

4. Modeling Platform

Modeling: A New Software Development Discipline?

The first step in the traditional development process involves business analysts and the business department working together to draft the requirements for the software to be developed. Then, they describe the requirements in prose and give them to the development team. We still use this traditional requirements analysis form for projects that are based on A12 – albeit to a lesser extent. But there is also another role: business analysts and experts can use modeling tools to design and adjust large parts of the application independently. They get much more creative leeway and **become Co-Developers/Citizen Developers**. The adjacent figure illustrates the differences between the two approaches.

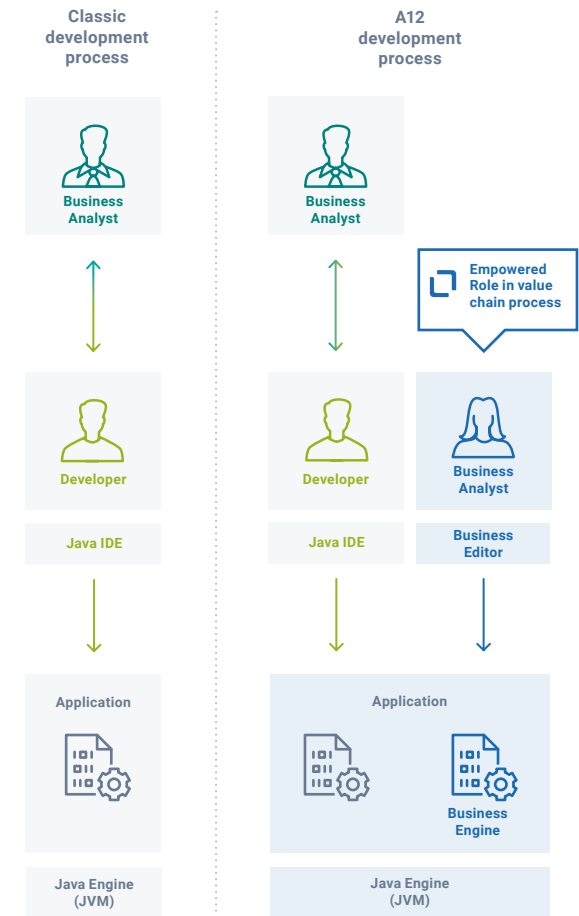
In most projects, mgm provides business analysts as part of the development team. This is beneficial as they are already familiar with the modeling tools and techniques. Customer-side business experts are usually involved from the beginning of the project. After an introduction to the modeling tools, they are then in a position to adjust essential parts of the application on their own.

The Modeling Concept of A12

The modeling approach of A12 differs in one essential point from the modeling approaches of other low code platforms: A12 follows the “data first” modeling paradigm.

Instead of starting with clicking together a user interface, A12 modeling starts with the definition of business relationships.

The decisive advantage of this approach is the clear separation of the domain description from a specific application. This creates synergies through the cross-application and cross-context use of domain expertise and great flexibility for the further development and maintenance of long-lived enterprise software.



The classic role allocation is shown on the left. Role allocation in the model-based approach is shown on the right. The business analyst helps the developer by independently designing parts of the application.

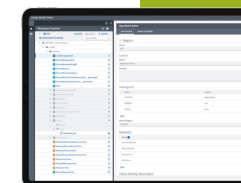
4. Modeling Platform

Main Modeling Dimensions



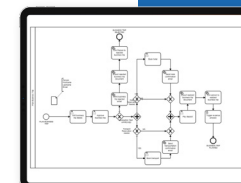
App modeling

- Application framework



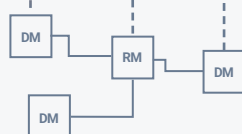
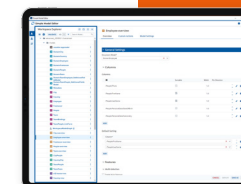
Workflow modeling

- Business processes based on BPMN 2.0 (Business Process Model and Notation)



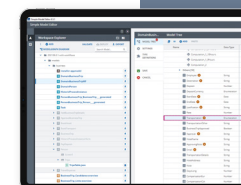
UI modeling

- Form models → Forms
- Overview models → Tables
- Tree models → Hierarchies
- Content models → Page content



Data modeling

- Document model
- Relationship model
- Mapping model
- Rule language



4. Modeling Platform

Modeling Business Expertise and Application Logic

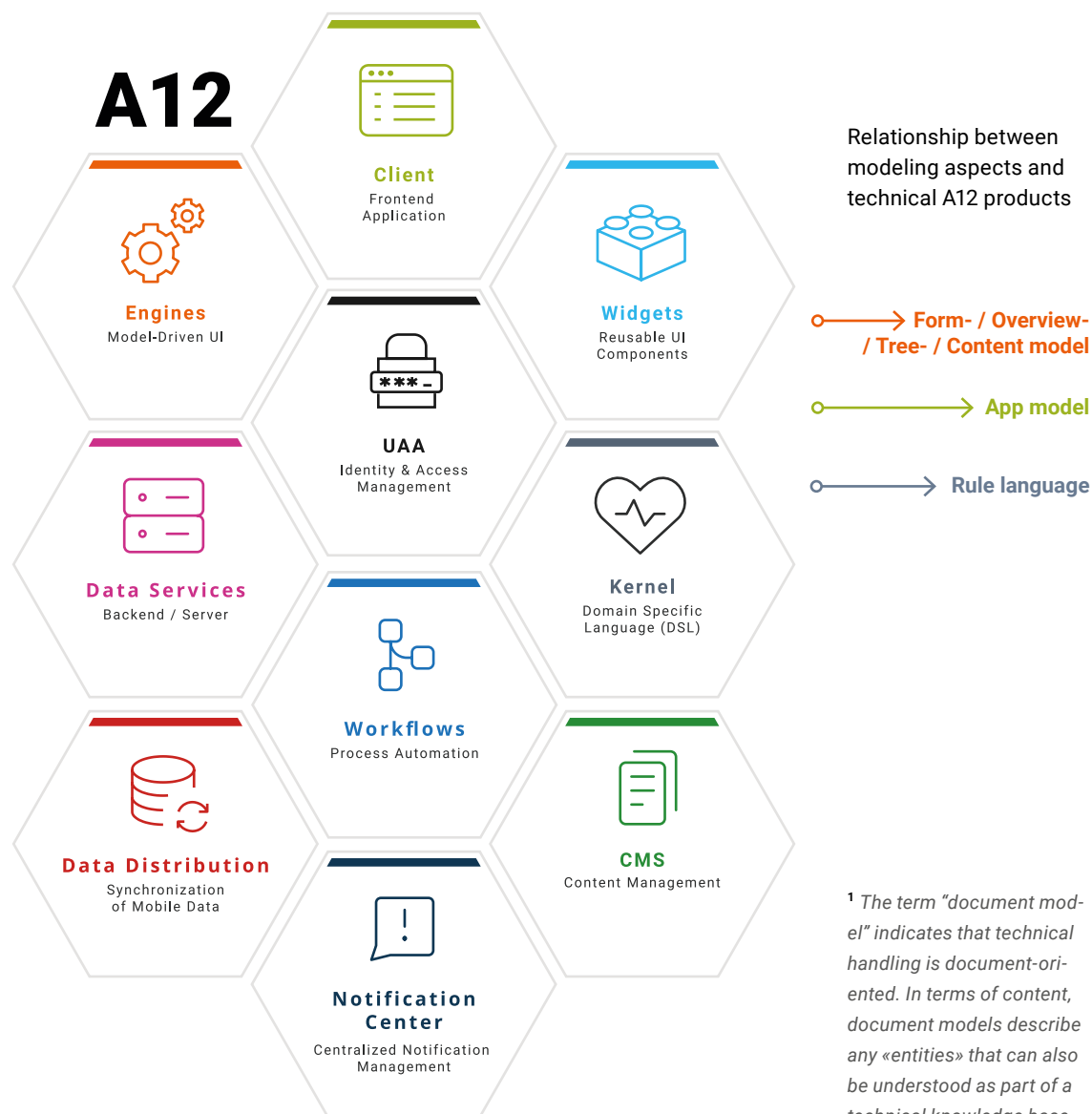
Defining the enterprise entities and their relationships to each other in **data models** is an essential modeling task. Business analysts and experts can use a modeling tool to create and adjust the data structures of mapped entities, such as contracts or products.

They can also use an integrated kernel language to define **validation rules** and **computations**, i.e. to map the application logic. **Relationship models** can be used to describe links between data models.

Modeling business-specific aspects keeps business expertise and technology separate from each other. Business content can be modified without any technical adjustments being needed. The technology can be developed further without all the business content having to be adjusted. We are confident that this separation of business expertise and technology will shape the future of enterprise software development. It accelerates development, prevents costly reimplementation and makes it possible to adapt to changes rapidly.



The rule language for validations and calculations is implemented in the technical A12 component Kernel
→ see also p. 50.



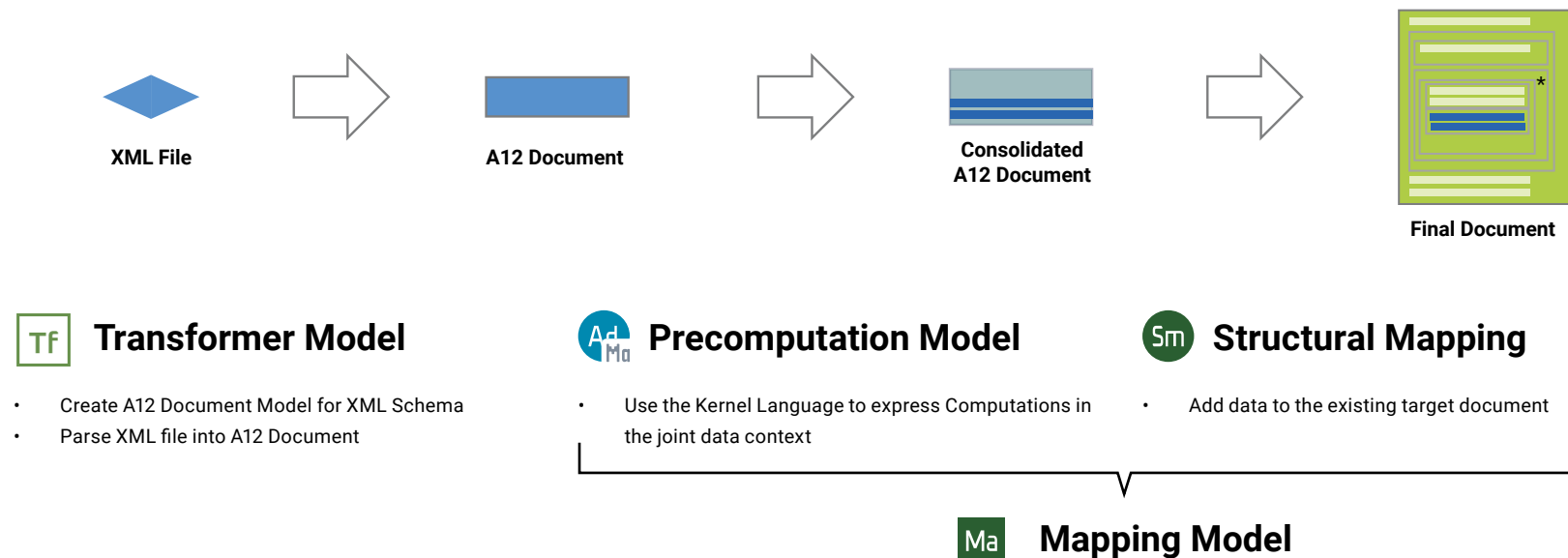
4. Modeling Platform

Modeling Data Transformations and Mappings

Data transformation and mapping are essential wherever data from different sources is brought together – whether integrating a business application with a neighboring system or in large-scale data consolidation initiatives such as register modernization in Germany. Until now, the implementation of these operations were considered technical tasks and needed to be programmed by a development team. With the transformation and mapping features in A12, however, these operations can be modeled by subject matter experts. They usually have a much better knowledge of the source data, the intended data and the user interactions. With the editors present in the A12 integrated modeling environment,

the Simple Model Editor (SME), modelers can express the business requirements directly in the models. The A12 Mapping Model allows to select the target and different A12 Document Models as sources. Necessary data consolidations can be expressed as Computations in the powerful Kernel Language using the whole data context of the joined sources and the target documents. In the next step, the data is transferred to the target document. Because those steps are expressed in separate models, they can be plugged together in different ways. This gives optimal flexibility to adjust to ever-changing source and target data schemes.

To complete the data pipeline, A12 also offers the A12 Transformer that converts XML data to A12 documents and vice versa. It allows to seamlessly integrate to external APIs in both directions. The Transformer Model can currently be integrated via a command line tool into the CI/CD-pipeline; but integration into the SME is also planned for the near future.

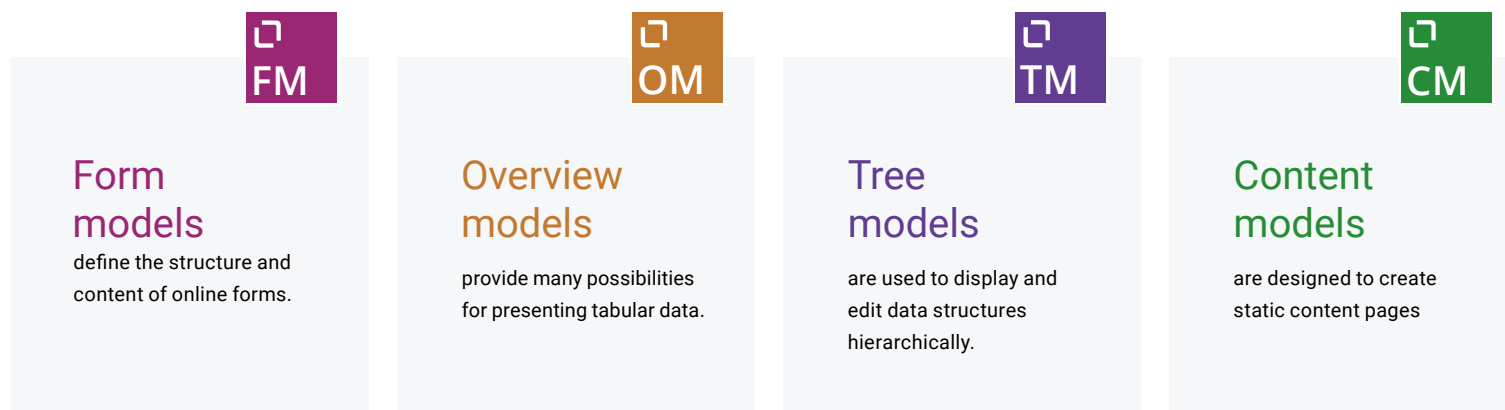


4. Modeling Platform

Modeling of User Interfaces

Based on the data models, business analysts are able to create specific parts of the user interfaces using A12's modeling tools.

The modeling of the user interfaces is currently limited to the areas in which model-driven components are used. A number of special UI models are available for this purpose:



The modeling does not follow a what-you-see-is-what-you-get principle. Instead, the models describe the underlying structures of the user interface. This has the advantage that the models are again independent of the technical implementation. The plasma design system is used for the actual representation. It ensures a coherent representation and a coherent user experience.

UI models usually refer to A12 data models. They establish connections between the fields of data models and UI elements. Let's take an input field as an example: A UI model describes its position in a form, its label, and possibly additional user instructions in a text field. A data model specifies the underlying data type and validation rules.

For each UI model, A12 provides a corresponding engine - the Form Engine, the Overview Engine, the Tree Engine and the Content Engine. They bring the models to life in an application.

→ Read more on p. 48.

4. Modeling Platform

Modeling Workflows

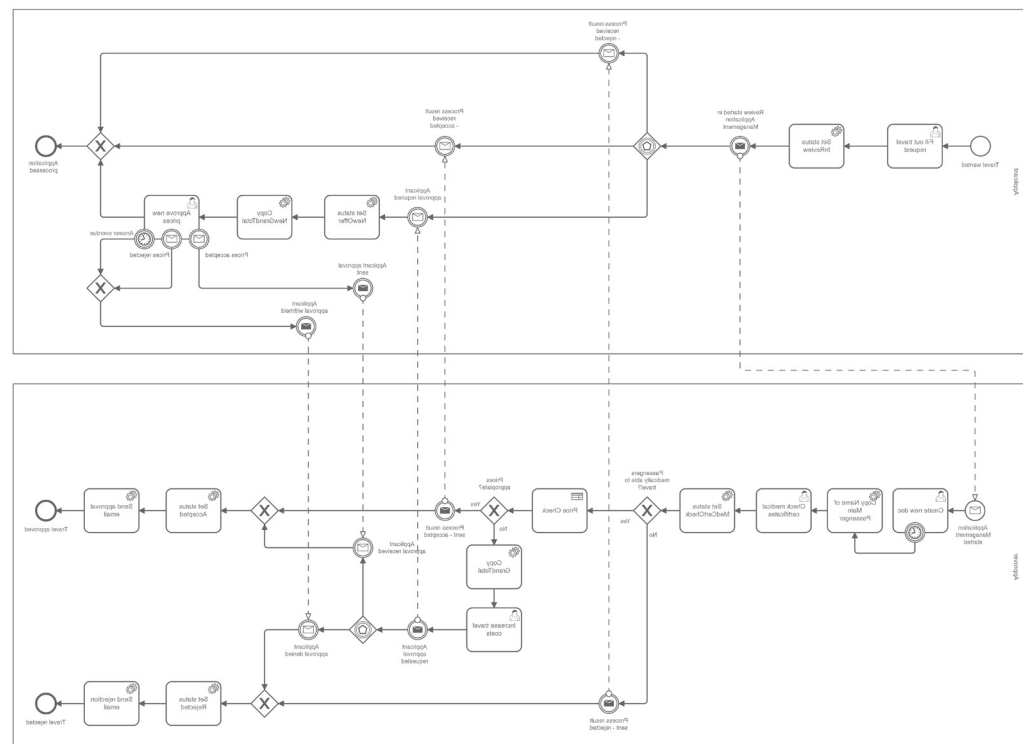
For modeling business processes, A12 relies on BPMN 2.0 (Business Process Model and Notation), an existing, established standard. The modeled business processes fit seamlessly into the modeling concept of A12. Document models describe the data used by a process. With the help of form models, the respective user tasks can be implemented in detail.

Modeling the Structure of an Application

The framework of an application can be defined with an **App Model**. It acts as a kind of container for all other models.



The app model offers configurations for certain functionalities of the technical component **Client** (see p. 49).



Modeling Print Templates

In the context of business applications, the challenge of generating PDF documents arises again and again - whether it is a contract in the insurance environment, a proof of invoice in an online marketplace or the notification of a government service. With A12's Print Model Editor which is seamlessly integrated in the SME, print models can be created and edited and easily brought into A12 applications." The resulting

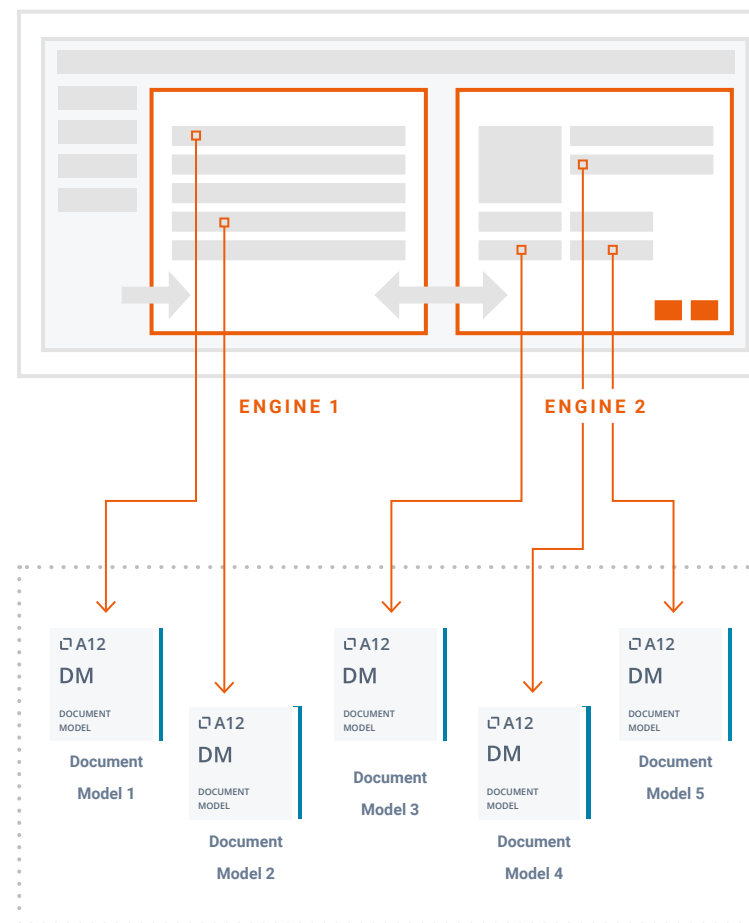
PDFs are compliant with the PDF/A-3 and PDF/UA standards and therefore accessible. The editor enables the convenient design of segments and sections by adding and placing individual elements such as text and images using drag & drop. Fields, calculations and tables can also be inserted directly, which refer to selected A12 document models and are filled with the corresponding stored data.

4. Modeling Platform

More Complex Modeling: Composed Document Models

With Composed Document Models (CDMs) it is possible to **use several document models in one engine** - provided that there is a relationship between the models (defined in a Relationship Model). Thanks to CDMs, a form can be fed with data defined in completely different models. The modeling concept of A12 gains significantly in flexibility and expressiveness through CDMs.

The SME offers a dedicated editing area for creating CDMs. CDM-based forms, into which fields from different document models can be inserted, can also be modeled directly in SME. In addition to cross-model validation and calculation rules, CDM support also extends to workflows. For example, composed data can be referenced in process models, and fields in task lists can originate from different document models.



4. Modeling Platform

The Model Types of A12

CATEGORY	NAME	DESCRIPTION
Data Model	Document Model	A12 document models contain field definitions and associated validation rules in a hierarchy of groups. Validation rules range from simple constraints - e.g., the definition of mandatory fields - to complex patterns and conditions across multiple fields.
	Relationship Model	Relationship models describe links between documents. They model the relationship properties and constraints.
Data Processing	Mapping Model	The Mapping Model allows you to convert an input data structure into a target format—from simple field mapping and filter operations to complex calculations.
UI Model	Form Model	Form models define the structures and contents of online forms. A12 forms consist of common UI elements such as input fields, buttons, labels, checkboxes, etc. The modeling tools provide powerful ways to organize these elements.
	Overview Model	Overview models offer various possibilities for tabular presentation of data.
	Tree Model	Tree models allow data structures to be displayed and edited hierarchically.
	Content Model	Content models are designed to create static content pages - from the start and information page to the imprint.
Workflow	BPMN 2.0	A12 supports modeling of business processes in the BPMN (Business Process Model and Notation) standard. BPMN models interact seamlessly with A12 models.
App Model	App Model	An app model defines the framework of the application and acts as a kind of container for all other models.
Output Model	Print Model	The Print Model functions as a template for PDF-representations of inserted A12 Documents.

4. Modeling Platform

Advantages of “Data First” Modeling With the A12 Rule Language

Business experts and analysts can create and modify domain-specific models for enterprise applications using A12's data modeling tools. No programming knowledge necessary! Data models encapsulate the central aspects of the enterprise logic. They describe the entities with which enterprise applications operate, such as contracts and products with all their properties.

The use of data models has several advantages:

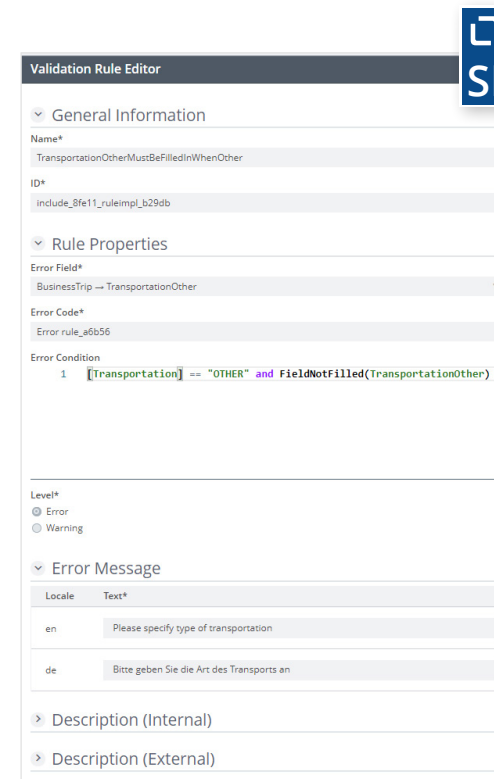
- ⊕ Reduced development costs and customisable applications
- ⊕ Business experts can modify the applications on their own. Developers are not needed to rework the implementation every single time there is a change in the enterprise domain.
- ⊕ The explicit direct storage in models makes it possible to search for and research business expertise. This also provides, for example, explicit traceability of business changes.
- ⊕ Improved reusability and independence from technologies

An important part of data modeling in A12 is the **rule language for validations and computations**. Based on business requirements, it enables the definition of rules that cover all conceivable field-related validation tasks. The most comprehensive data validation possible is crucial to avoid security risks and to ensure data integrity in business applications.

The language contains many predefined predicates. It supports nested comparisons, arithmetic operations and provides special operators for handling elements like dates. It also supports special conditions for checking in which configuration fields may or may not be specified. The various subconditions and operations can be combined.

The modeling tools for document models support the language directly. It has been successfully deployed for years in large productive software systems. Our customers use it in many projects to independently manage validation rules and computations.

The language combines the simplicity of the propositional logic with the expressiveness of the predicate logic. It is particularly well suited for forms and strong typing in business domains.



Creating a rule in SME

4. Modeling Platform

The rule language has the following key features:

- ⊕ Rule conditions describe errors – the end-user is thus shown messages related to the specific error scenario
- ⊕ Use of logical connectives 'And' and 'Or' to combine different subconditions
- ⊕ Negation operations are not used. Instead, the different predefined conditions are each provided in positive and negative form. This ensures that the subconditions are simpler and are compiled in a more uniformly structured way. This makes the rule conditions more readable and clearer.
- ⊕ Predicate logic quantifiers are not provided as formally logical parts of the language but implicitly via operations. This ensures that the conditions are based on an expert's formulations and are therefore easier to understand.
- ⊕ The rule languages' logic operations allow the tree and repetitive structures to be queried directly
- ⊕ Supports set and filter operations on tree structures and repetitive structures, e.g. "add up all capital gains from all equity funds"
- ⊕ Facilitate iterations via repetitive structures and shorten the control conditions
- ⊕ Computations and validations based on the same language, so the full validation language can also be used for computation preconditions. All of the language's set and filter operations are available for formulating the computation operations and values can be computed for all predefined field types

Features:

- ☑ A powerful and versatile validation and computation language
- ☑ Auto-completion and syntax highlighting
- ☑ Predefined predicates for fields, lists of fields and groups that can be combined freely
- ☑ Arithmetic operations, comparisons, special operators for processing

MODELABLE WITH A12

INDIVIDUALLY REALIZABLE

Domain expertise - data models with validation rules and calculations

complex algorithms (e.g. generic premium calculator in the insurance environment)

Frame of an application including placement of model-driven engines

placement of simple widgets

Forms, including repeatable structures

definition or adaptation of design elements

Tabular overviews of data sets

Tree-like overviews of data sets

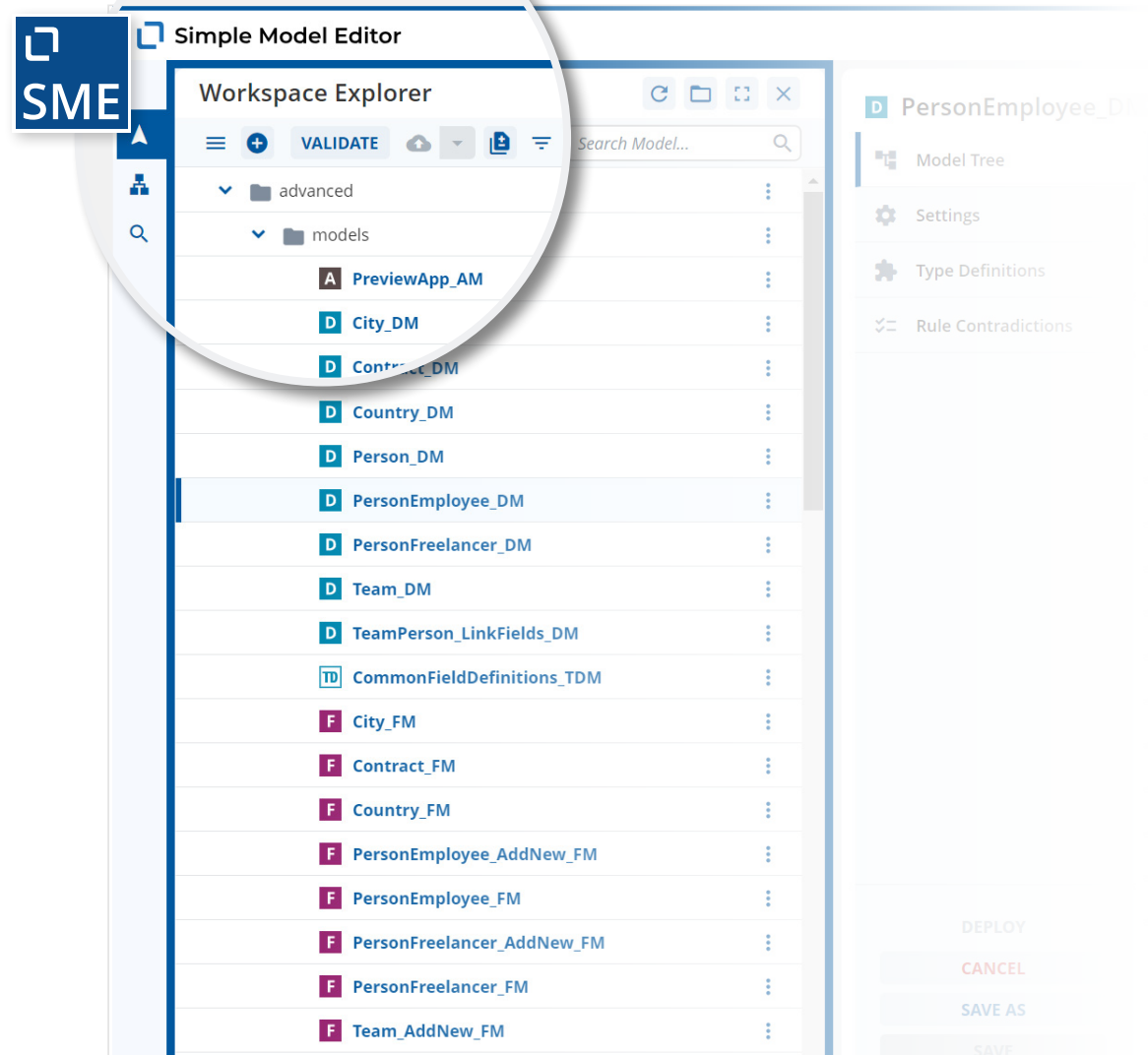
Relationships between different model-driven components

Workflows following the BPMN 2.0 standard

4. Modeling Platform

Simple Model Editor (SME)

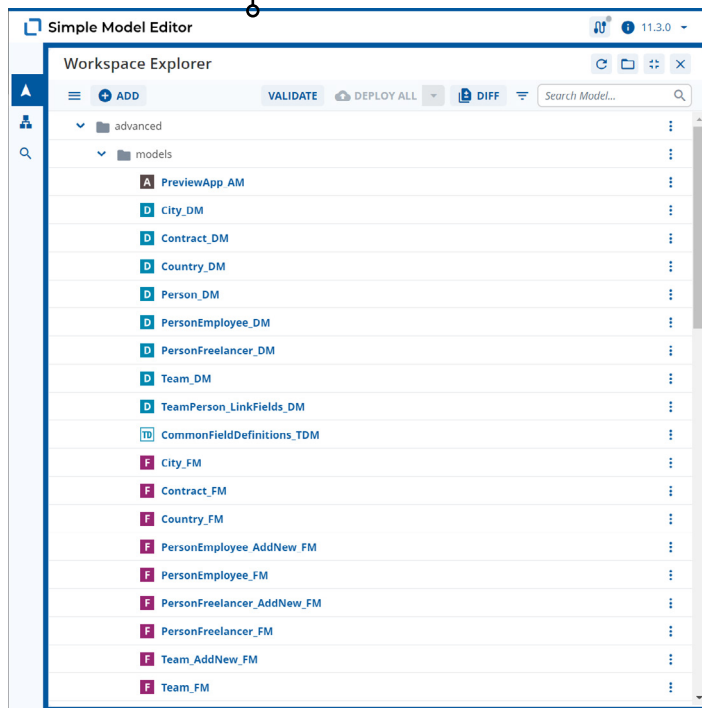
The Simple Model Editor (SME) is the control center for modeling in A12. The tool enables business analysts and domain experts to design and test key parts of business applications themselves without any programming knowledge. A special feature: The SME was built as a tool for A12 itself with A12.



4. Modeling Platform

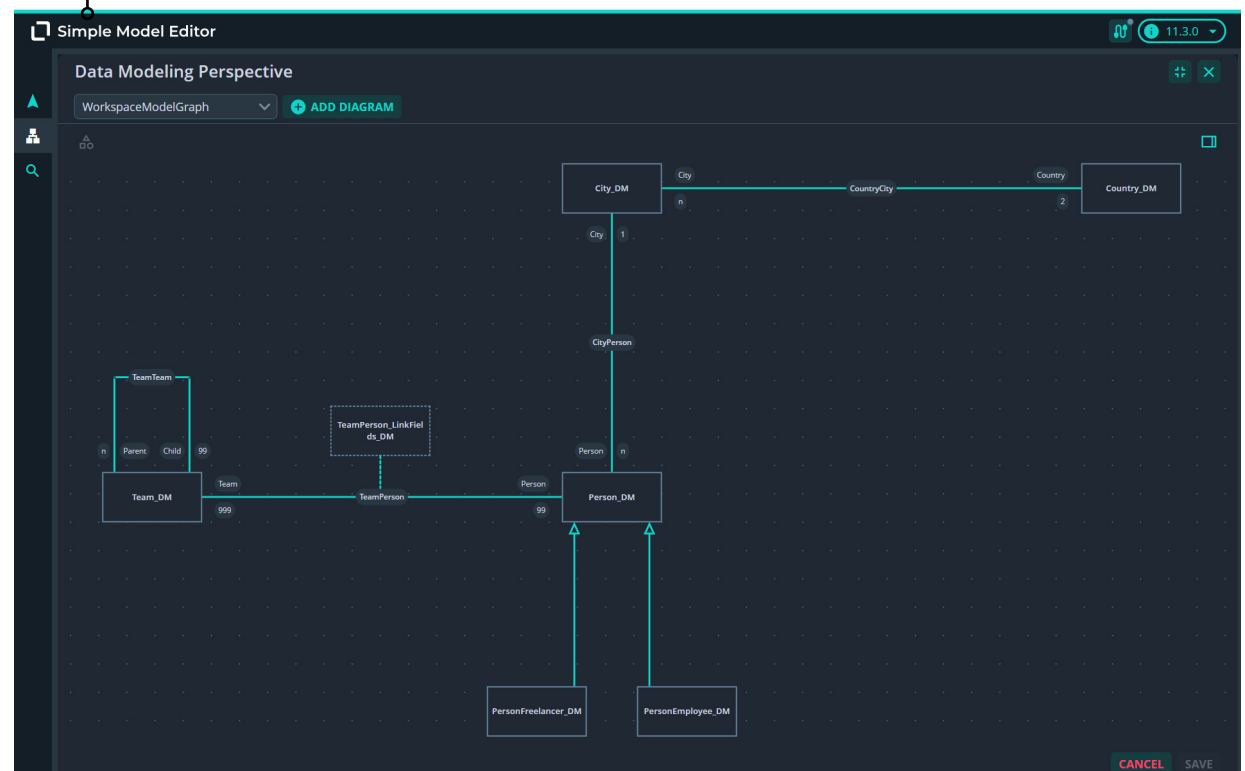
Manage models within a project workspace

- ⊕ Easy-to-use Workspace Explorer for managing all A12 models
- ⊕ Create, delete, rename, move models
- ⊕ Support for model refactoring



Model the domain-specific content of business applications

- ⊕ Define data models for entities and relationships
- ⊕ Descriptive perspectives: Model trees and field editors
- ⊕ Visual perspectives: Model Graph Diagram Editor
- ⊕ Rule editor for validations and calculations with autocomplete



Design online forms

- ⊕ Quickly create forms by dragging and dropping data fields
- ⊕ Live-Preview: Changes are immediately visible
- ⊕ Repeatable structures for data entries with uncertain quantities

4. Modeling Platform

Create tables and tree-structures

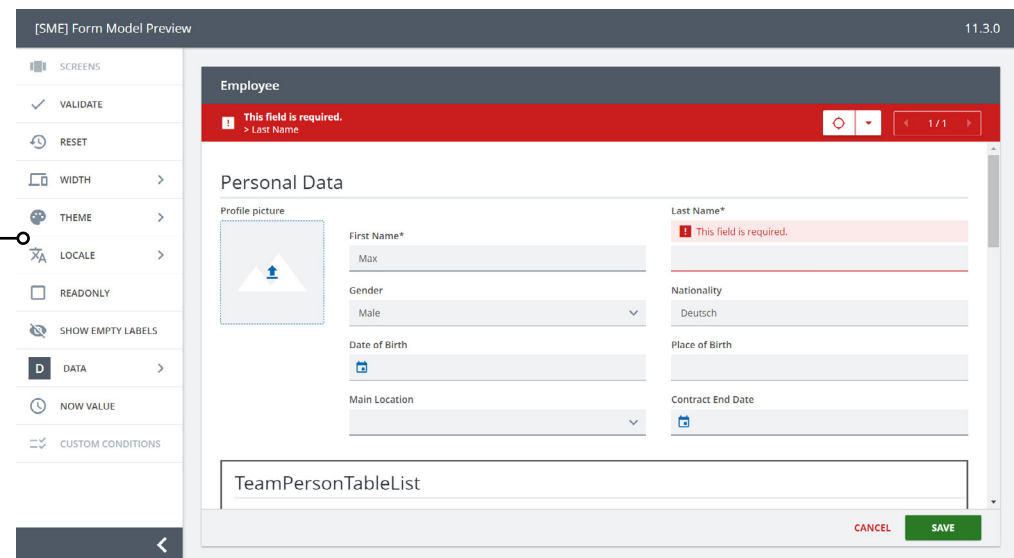
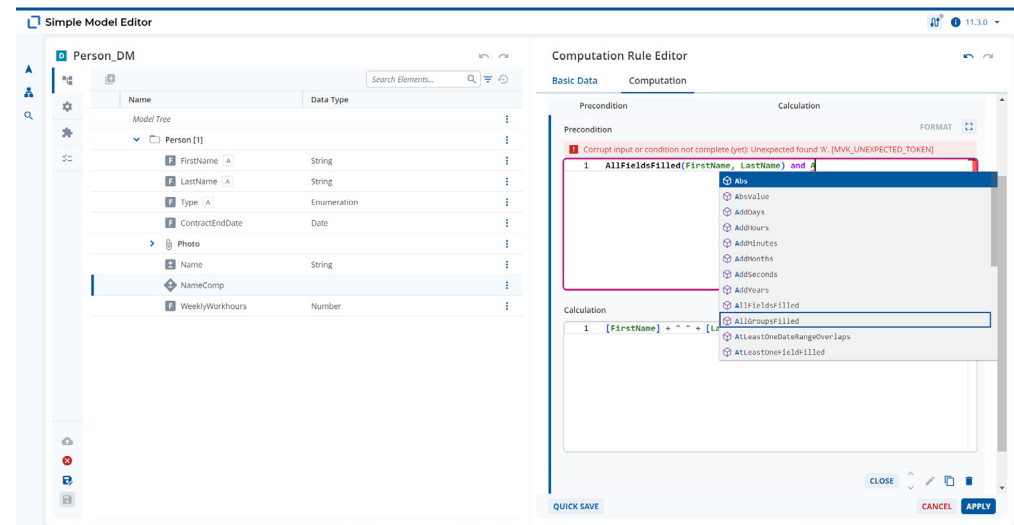
- ⊕ Modeling of data-driven overview tables and tree-structures
- ⊕ Easy-to-use features such as multi-selection, full-text search, filters, etc.
- ⊕ Flexibly configurable buttons for custom actions

Model application frames

- ⊕ Global layout settings
- ⊕ Definition of sub-regions such as content area and sidebar
- ⊕ Manage modules for essential areas of the application

Quality assurance: testing modeled content

- ⊕ *Ad-Hoc-Tests* for targeted testing of selected validations and calculations
- ⊕ *Rule Contradictions Report* uncovers contradictions in rules that span multiple data fields
- ⊕ Automatic detection of consistency problems in models



4. Modeling Platform

Provision of the Modeling Environment

The A12 modeling tools were developed to be web-based from the very start. They can be installed locally or obtained as software as a service in the cloud version.

Installer: Using Modeling Tools Locally

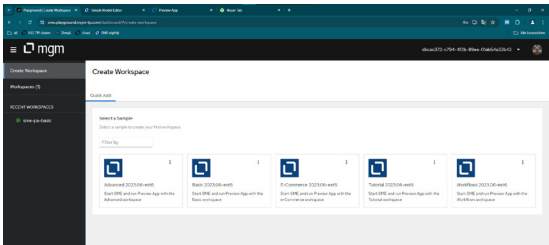
To be able to use the modeling tools of A12 locally, the **A12 Installer** is available. It bundles all relevant tools in one installation file. The installer is provided with each release of A12 for Windows 10, macOS and Ubuntu Linux.

A set of included sample apps makes it easy to get started and provides a starting point for your own modeled apps. With the help of the included **Preview App Control**, modeled programs (preview apps) can be executed locally in the browser.

PARTS OF THE INSTALLER	DESCRIPTION
Simple Model Editor (SME)	Modular tool that bundles numerous modeling functionalities of A12
Camunda Modeler	Tool for modeling workflows
Preview App Control	Application for running A12 applications in the browser
Workspaces	Sample applications (preview apps) that demonstrate the modeling scope
Documentation	Reference to existing online documentation, which can optionally be installed locally as well

Cloud-based modeling environment

The A12 modeling environment is also available as Software-as-a-Service (SaaS) from the cloud - and can be used without any installation. It can be accessed directly via the browser. Once users have logged in on the start page of the cloud offering, they are greeted by an administration interface - the *Cloud Modeling Control*. New *Cloud Modeling Environment workspaces* (CME workspaces) can be created here and existing ones can be started. For new CME workspaces, the familiar 5 sample workspaces from the local modeling environment are available as a basis. After starting a work-



space, the Simple Model Editor (SME) opens in a new tab and the Preview Application is available. The SME officially supports the Chrome (Windows, MacOS), Edge (Windows) and Chromium (Ubuntu) browsers.

05

Runtime Platform

The A12 runtime platform consists of a set of modular client and server side components in a modern enterprise architecture. It provides robust components for typical enterprise application requirements. At the same time, it gives the development team full control through fine-grained entry points to plug in their own code and implement individual project requirements.

5. Runtime Platform

A Different Range of Tasks for Developers

The model-driven approach also comes with a variety of changes for developers, too. They are no longer solely responsible for building the whole application. Their workload is smaller, especially in relation to handling business changes. The application can be compared to a play; the models designed by the business analysts are like the protagonists in the limelight. The developers, however, make it possible for the play to be performed at all. They prepare the stage and make sure that the protagonists are shown in the best light.

Modeled business expertise reduces workload

In a conventional software project, the development team is responsible for coding the whole application on their own. To do this, the team must understand the idea behind the application down to the smallest detail. But that's a massive challenge for highly complex application fields, such as taxation or industrial insurance.

The model-driven approach changes this situation. Business analysts and experts map the business logic in models and put them directly into the software.

This greatly lessens the developers' load. They no longer need to understand the modeled business aspects nor implement it by hand. The focus of the work shifts.

Connecting, maintaining and extending the application platform

Projects based on A12 do not start off as greenfield projects. They build on an existing foundation. This foundation isn't static; it's being constantly developed. One of the main things that the developers have to do is to connect the foundation (the project's A12 application platform), maintain it and, if necessary, extend it individually. The Technical Professional Services Team provides support.

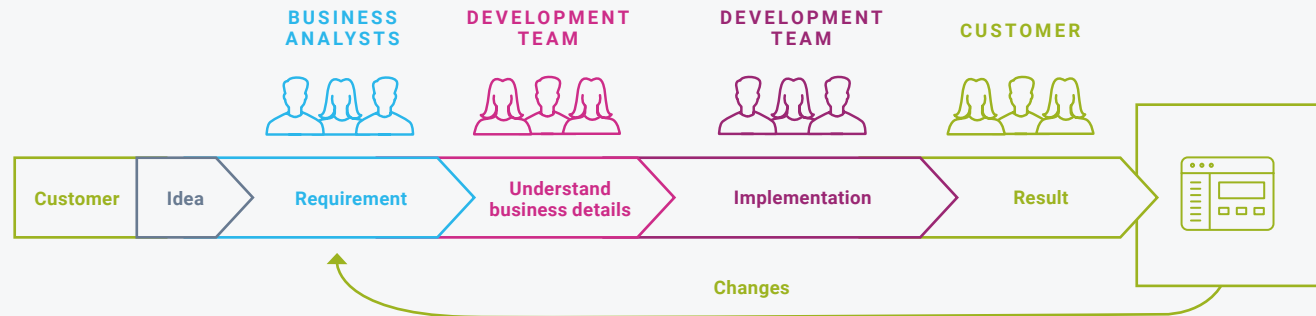
More complex functions and integration work

Developers also write code that implements more complex functions. An example of this is a complex computation that goes beyond the existing scope of the modelling tools. Furthermore, one task still left to the developers is to integrate the application into the existing heterogeneous IT landscape.

5. Runtime Platform

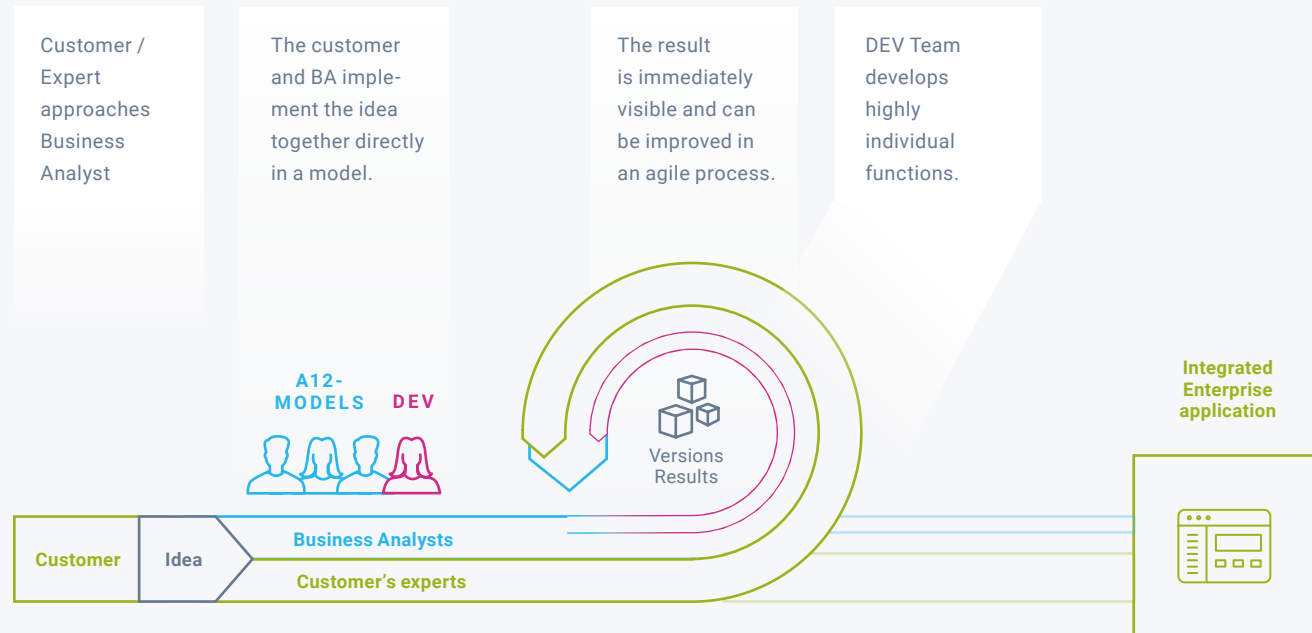
Traditional Approach vs. Model-Based Approach

Traditional approach



Model-based approach

- ✓ Requirements in Models
- ✓ Develop together
- ✓ Result visible immediately
- ✓ Agile and fast



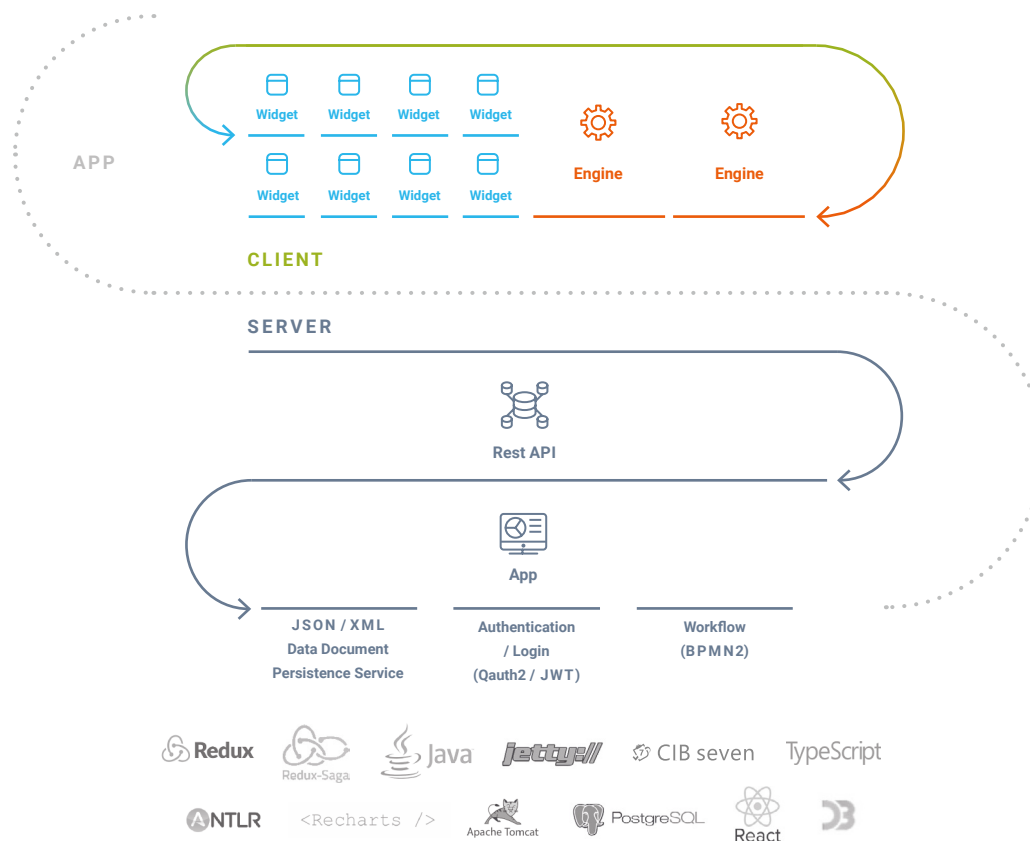
5. Runtime Platform

Architecture

The development process of business applications is continuously shaped by technological changes. A12 meets these challenges and offers a runtime platform for modern, web-based business applications. Starting from a robust core and modular solution modules, we continuously advance this platform on all levels. For this purpose, we adapt new technologies and paradigms, as long as they contribute to the goal of making the development of high-quality business applications easier, more efficient and more sustainable.

We benefit from an important capability of the low-code approach: many of the most complex and important aspects of the application are modeled in A12 and can thus be expressed in a largely technology-neutral way. In fact, even complex forms solutions survive the technology shift from JSPs and XForms (2012 and earlier) to Angular (circa 2015) to React (2017 and later). The necessary foundations - the UI engines as runtime interpreters of models - change, but the models remain.

The **A12 Client Framework** addresses the complexity and challenges of modern web applications using the single-page application (SPA) approach. It is also the basis for quickly building modularized frontends (Microfrontends). It leverages the modern and proven React/Redux technology stack, integrates A12 UI components such as Engines and Widgets, and interacts with A12 backend services such as A12 Data Services and Workflows using REST APIs. Data and models are JSON data documents. Custom backends can be easily connected, just as overall most aspects of the A12 client framework can be customized or even overridden through extension points. The **server-side A12 services** provide, among others, the data services for the aspects of data storage, search, and model repository, as well as workflows (Camunda/BPMN 2), authentication/login (LDAP, SAML,



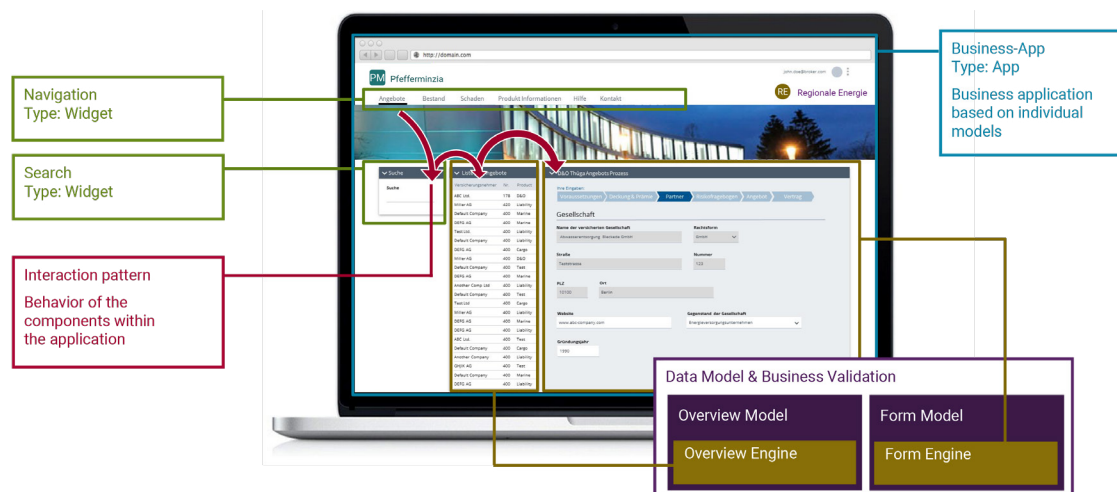
OpenID Connect, OAuth 2, JWT), and user/role management. The services are built on Spring Boot and can be used out-of-the-box, but also easily extended with customer-specific code. Behind this are supporting open source products, in-

cluding Postgres as database, Keycloak for access management and single sign-on, and CIB Seven as workflow engine.

5. Runtime Platform

Document-Oriented Data Access and Model Graph

The A12 architecture is based on the concept of hierarchical collections of field values in JSON documents (Documents for short). Clients can access and store these Documents. Document models (schemas) specify not only field types, but also validation/integrity rules and computations in our highly expressive kernel DSL (Domain-Specific Language). These rules are automatically evaluated by the Form Engine during form processing, for example. The Query API of the A12 Data Services component allows you to filter queries for A12 documents, links, and data graphs very precisely and with minimal data usage. The query results can be sorted, paginated, and restricted to specific fields.



- ⊕ **Relationships** between Documents are fully supported; Documents can be linked and relationship properties and constraints can be modeled and are enforced by A12 Data Services. Furthermore, there is an inheritance concept (Subtyping) for Document models. This allows more complex domains to be expressed as a graph of Document models; we call this the Model Graph. Our tree engine uses the model graph to represent linked documents in a tree view, for example.
- ⊕ Thanks to the mentioned **CDMs**, views on the model graph can be queried, analogous to GraphQL.
- ⊕ **Batches:** The A12 Data Services API provides a Batch REST endpoint for transactional bundling of multiple document operations, such as creating a new document with simultaneous linking to another document. There is also an operation to partially modify documents to reduce network traffic.

A12 Frontends

The A12 architecture places a strong emphasis on simplifying client-side application development. It provides a field-proven application framework provided by the A12 Client Framework, Engines for working with models, and Widgets for reusable UI components.

The application framework uses an Application Model to control the interaction of the Engines, such as in a Master/Detail context. Written in TypeScript, the A12 client framework is based on React and uses Redux for state management and caching.

The framework offers a variety of **integrations**: a data access abstraction “Data Provider” with built-in support for A12 Data Services, the connection of process engines with built-in support for CIB Seven/A12 workflows (such as task

lists), an A12 Data Distribution Client (data sync, offline capability), and notifications via the Notification Center.

In addition, the A12 Client Framework offers many useful and powerful features such as asynchronous flow control using Redux Saga, dirty handling and undo mechanisms, URL routing, a layout provider abstraction with responsive defaults for desktop and mobile devices, and localization.

An A12 Frontend Client can be modularized according to the Microfrontend pattern. For this purpose, we technically use “Module Federation” from Webpack and have developed an application module registry based on it, allowing dynamic integration of these modules, for example according to user roles.

5. Runtime Platform

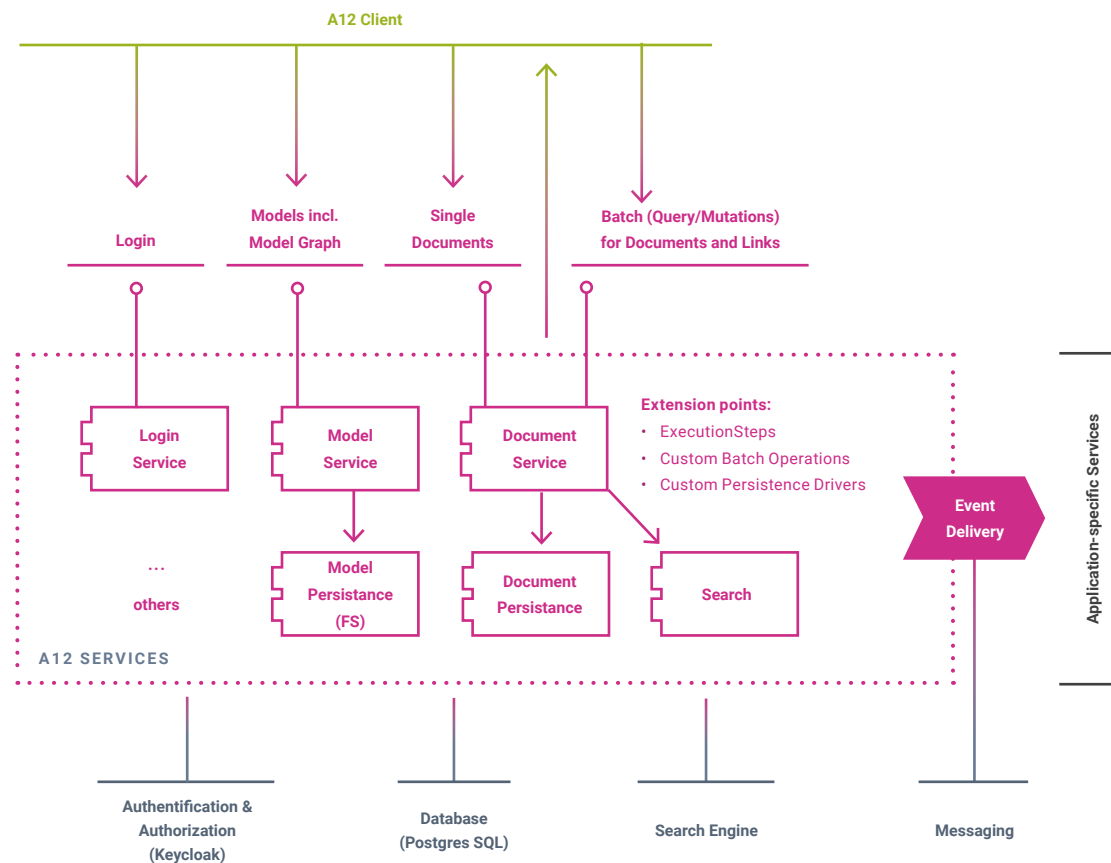
A12 Backend Services

The fundamental backend service is A12 Data Services. It provides access to models and Documents and also handles login with SSO support and optional Keycloak integration (LDAP, SAML, OpenId Connect, OAuth 2, JWT). The APIs are available as **stateless REST** endpoints and in Java. Persistence of Data Documents is supported by a set of reliable technologies such as PostgreSQL.

A12 Data Services, like all other server-side A12 services, leverages the **Spring Boot framework** and is available in three forms: as a standalone application, as a Spring Boot project for project-specific applications with custom code, or as a library to leverage selective features in existing Spring applications. In addition, there are numerous **extension points** and a comprehensive event system for easy integration of custom code handling before and after operations.

For scaling, the services can be operated in a Hazelcast cluster. Such a cluster can then be dynamically adapted to the load under Kubernetes. Our A12 Project Template already offers configurations for this.

Other server-side A12 services include **A12 Workflows** (based on CIB Seven/BPMN 2) and the **A12 User Management Service** with IDP support (Keycloak). There is also **A12 Data Distribution**, a highly scalable data distribution and sync solution with offline client capability. The Notification Service uses A12 Data Distribution for notification delivery.



The A12 Kernel is used on the client and server side. It validates data and computes derived data based on rules and field types described in data document schemas (called Document Models). Code generation ensures native code

for both client and server. On the Frontend, the rules and calculations are executed as native JavaScript, providing immediate feedback to the user during form processing.

5. Runtime Platform

Project Scenario for the Use of A12

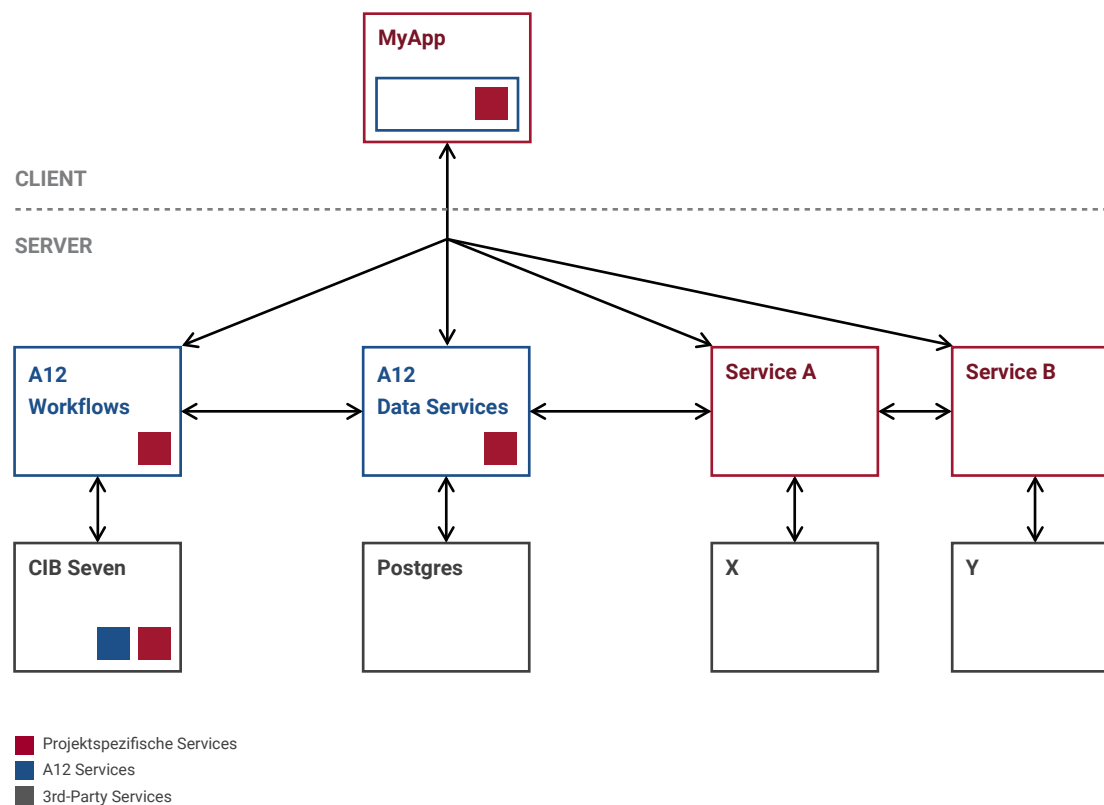
Thanks to its modular design, A12 can be used very flexibly and is also ideally suited for Microservice architectures and Microfrontends with extensive support in the framework. The following diagram demonstrates how the building blocks of A12 can interact with project-specific extensions and services as well as third-party components in a Microservice context:

About the Frontend: The resulting web application is dynamically composed of several parts and corresponding frontend projects: the application shell and two Microfrontends provided by the customer's own Microservices (A and B).

A12 components are used: the widgets and engines are customized and the A12 client framework is extended to meet the respective project requirements. For example, one can query data from the REST APIs of one's own microservices and prepare it as JSON documents via data provider abstraction, making it accessible to the engines.

The **server side** consists of

- A12 services with optional project-specific customizations and
- any project-specific services (e.g. as Microservice) with or without A12-specific extensions (e.g. the A12 Kernel as library or A12 Data Services as dependency).



5. Runtime Platform

Components

A12's runtime platform is modular and consists of a series of loosely interconnected components. Depending on the situation, they can be used flexibly in the project, even individually.

Most projects use the Client-Engine-Widget trio. Some projects use the back-end and server services provided by the Data Services module. Others write their own server depending on their requirements.



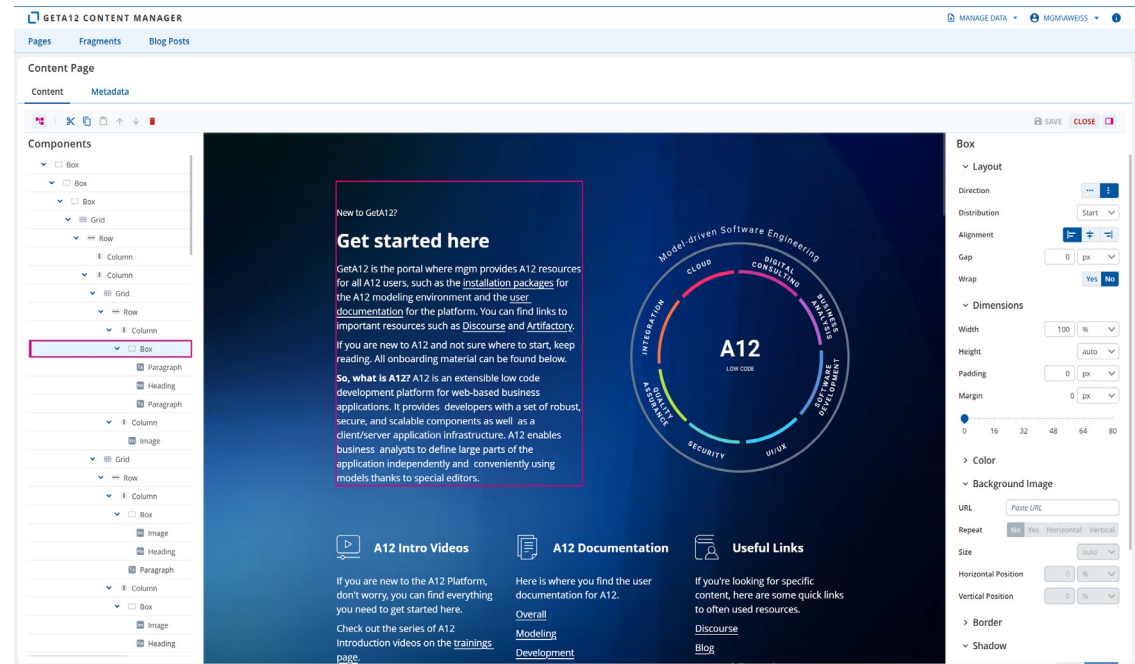
5. Runtime Platform

COMPONENT	ABBREVIATION	DESCRIPTION
Client	C	Model-driven, client-side runtime component. Implements the UI/UX concept of the Plasma Design System and supports desktop, tablet and smartphone. Main tasks are the orchestration of other UI components, especially the A12 engines, data retrieval and state management.
Engines	E	Model-driven UI components. Engines interpret data and UI models. They are based on the Plasma UI/UX concepts and use the widgets for rendering.
Widgets	W	Widget Library, based on Plasma UI/UX concepts. See also → A12 Widget Showcase .
Kernel	K	Bundles everything for the creation and processing of document models: modeling tools, language for validations and calculations, client- and server-side runtime components, Java and Typescript API.
Data Services	DS	API for managing models and data. It also contains routines for client/server communication, validation, persistence and indexing.
User Management, Authentication and Authorization	UAA	Bundles solutions around authentication (Keycloak, OAuth 2.0, SAML, LDAP), authorization (Spring Security, RBAC, ABAC, custom logic) and user management
Workflows	WF	Integration of Business Process Model and Notation (BPMN) in A12; enables graphical modeling of server-side workflows and their execution
Data Distribution	DD	Transport layer for synchronization of data
Notification Center	NC	Communication center for notifications such as tasks, appointments and reminders
Content Management System	CMS	Lightweight content management system

5. Runtime Platform

CMS Content Management System (CMS)

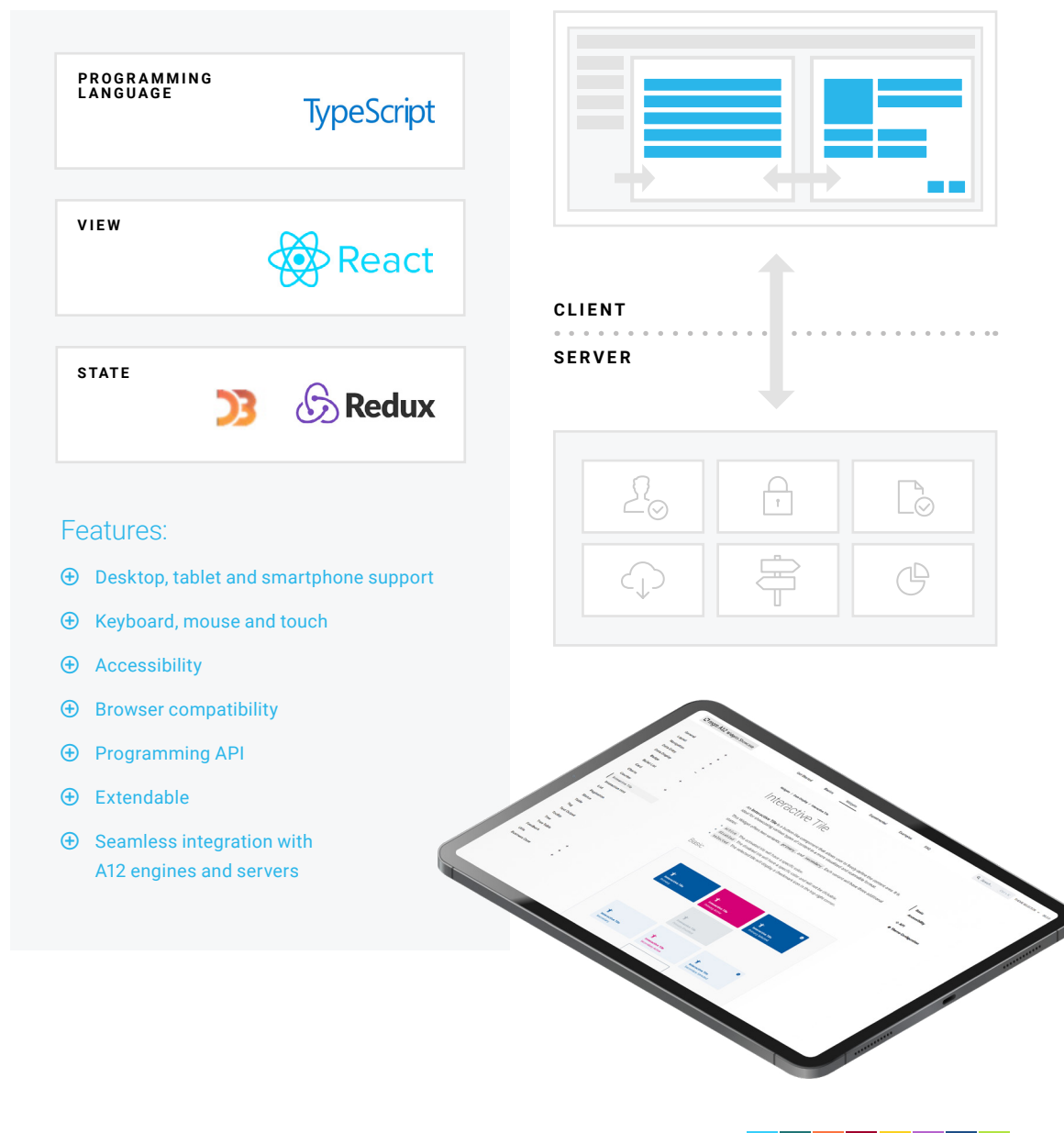
Modern web applications always include textual and visual content - be it in the form of an imprint for publicly accessible websites, a customized start page or a news and blog section. To manage such content, A12 offers a new, light-weight content management system (CMS). It provides a lean and integrated solution for A12 applications that contain editorial content. The CMS consists of an editor that can be used to manage and design pages and a content engine as a runtime component which interprets Content Models.



5. Runtime Platform

W Widgets

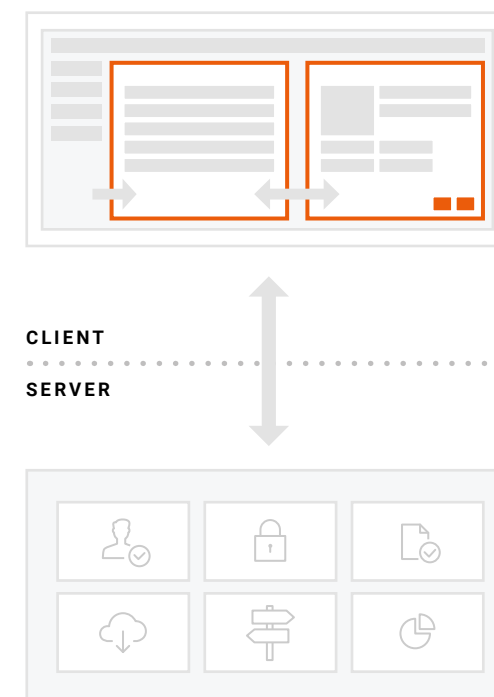
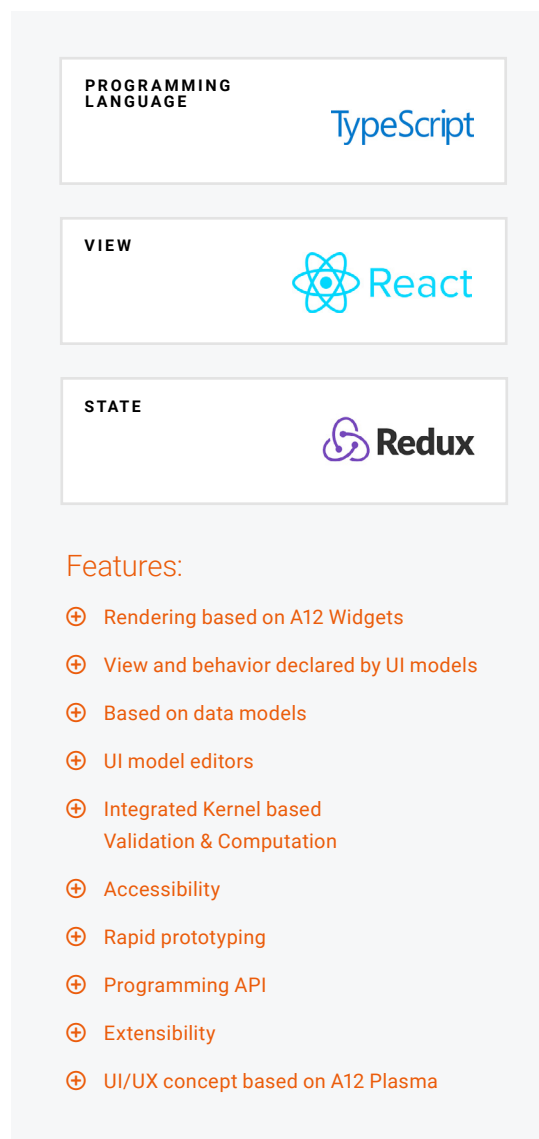
Widgets are reusable UI components that follow Plasma design conventions and UX concepts. They support enterprise applications that run on desktops, tablets and smartphones with keyboard, mouse and touch input. The components provide an easy-to-use, well documented, strongly typed API and are extensible and customisable.



5. Runtime Platform

E Engines

A12 engines are implemented in TypeScript. They are self-contained runtime components that interpret data and UI models. They are based on Plasma UI/UX concepts and use widgets for rendering.

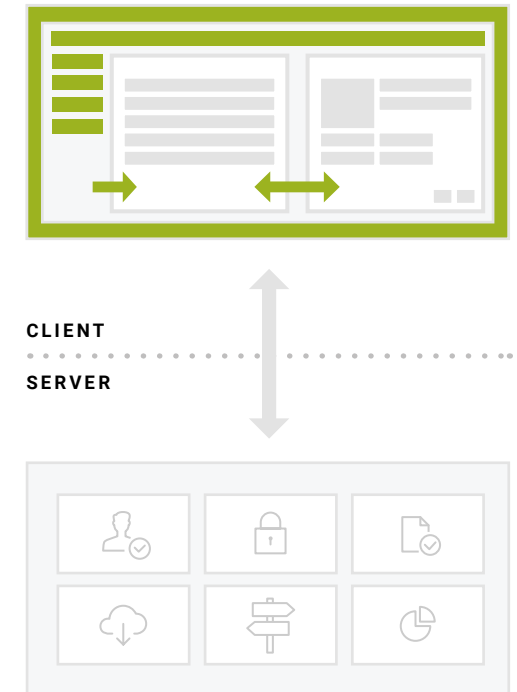
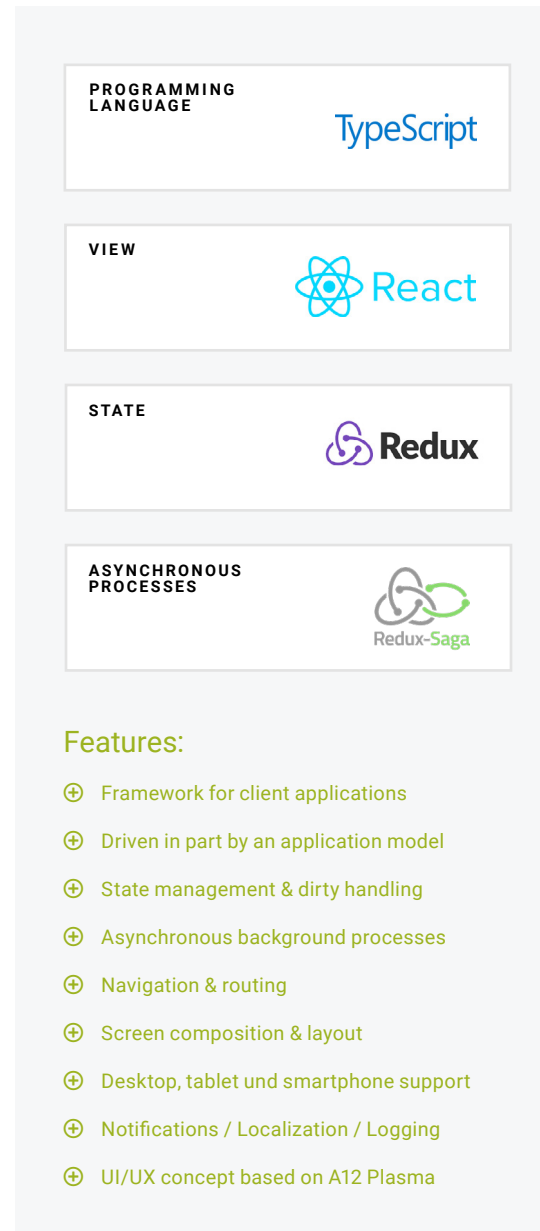


5. Runtime Platform

C Client

The model-driven, client-side runtime component makes it possible to declare the core aspects of the application, the modules, the navigation, the screens and the most important interaction patterns. Its main task is orchestrating other UI components, especially the A12 engines.

It also organises handling requests, data retrieval and processing, and status management. The client component implements the Plasma design system UI/UX concept and supports desktops, tablets and smartphones.

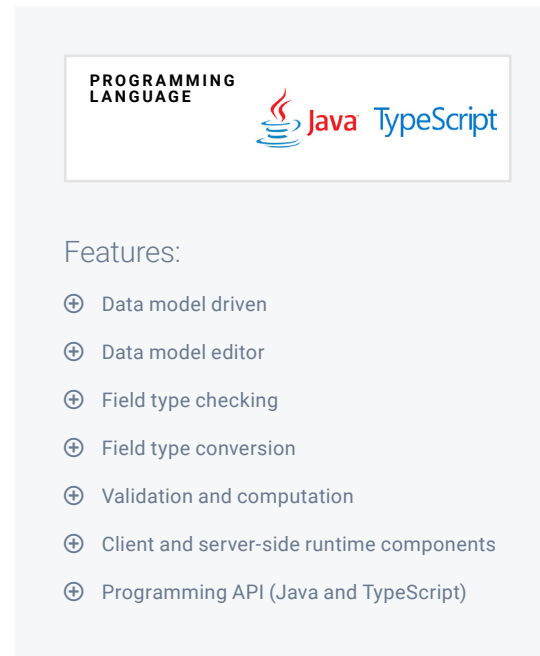


5. Runtime Platform

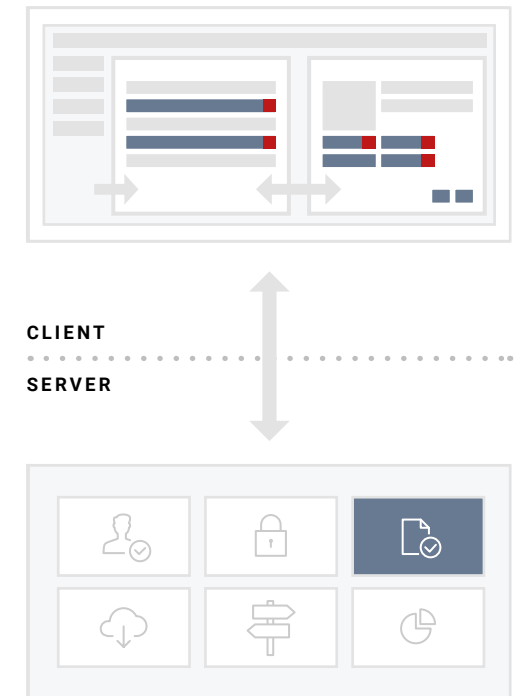
K Kernel

The kernel component bundles basic functions for creating and processing data models. Above all, it defines A12's domain-specific languages (DSL).

This includes all bases for validations and computations that are part of business modeling. The component includes client and server-side runtime components and a Java and TypeScript API.



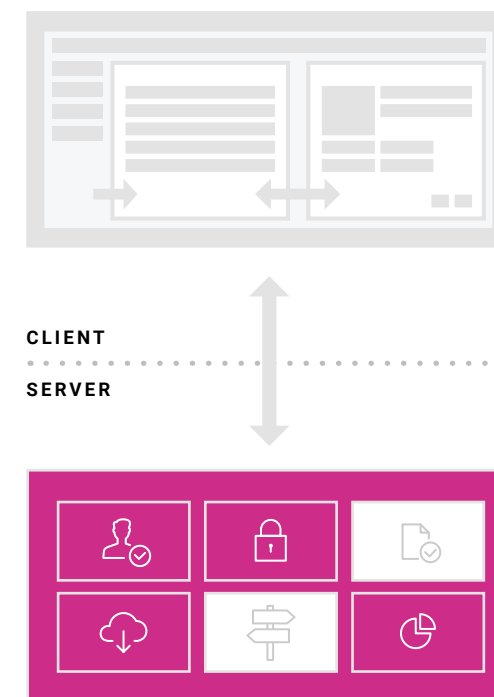
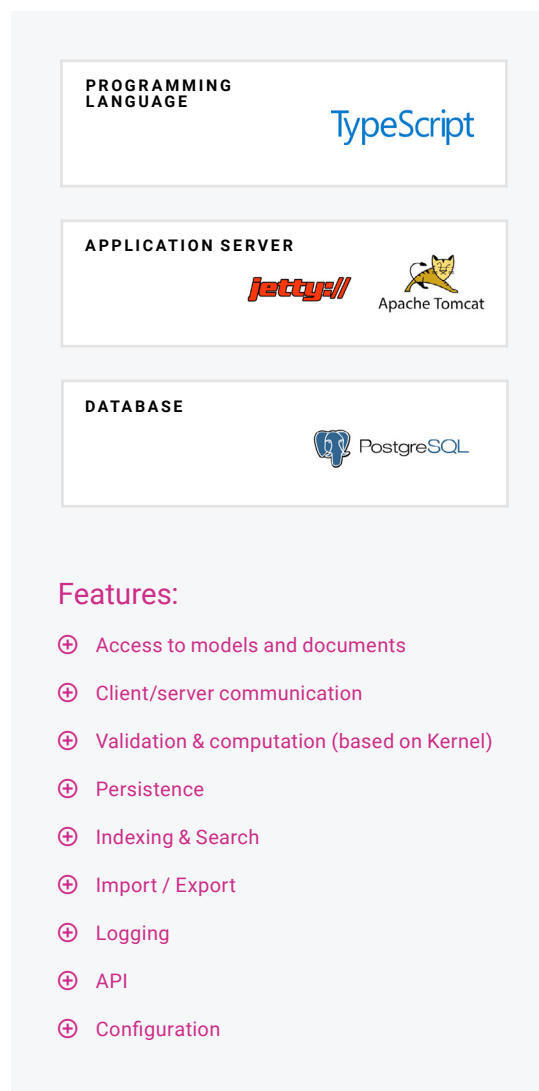
The kernel component includes the A12 DSLs, among other things



5. Runtime Platform

D Data Services

The Data Services component provides an API for managing models. It also includes routines for client/server communication, authentication, authorisation, validation, persistence and indexing. It is programmed in TypeScript for the client side and in Java for the client and server side.



5. Runtime Platform

UAA

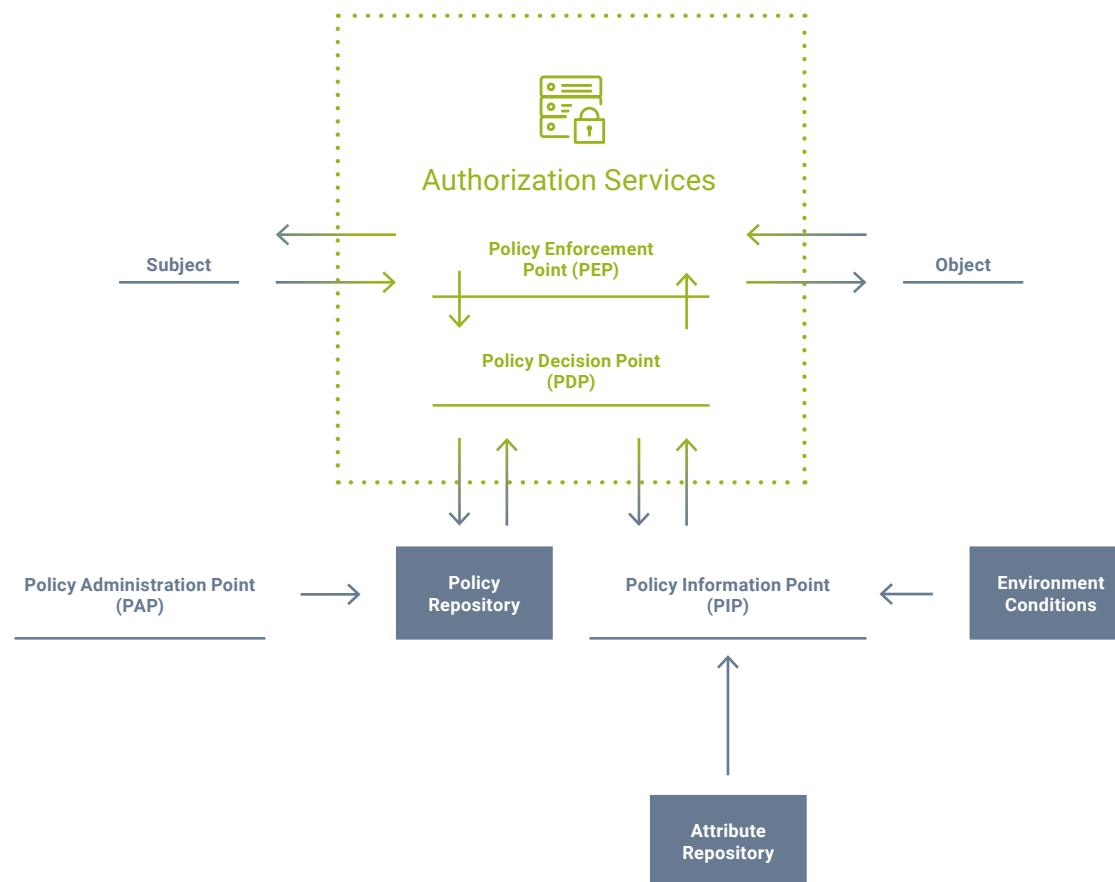
User Management, Authentication & Authorization (UAA)

A12 uses Keycloak, a tried-and-tested open-source solution, for authentication. It also supports both OAuth 2.0 with OpenID and SAML token-based SSO authentication and connection to LDAP.

A12's UAA components also provide a highly flexible and powerful authorisation solution, which can provide access rights in different levels of granularity. Both role-based and more complex, attribute-based rules can be specified, thus protecting access down to the field level of data documents.

The UAA component is supplied as a library. It can therefore be integrated both in the A12 server and in the application's standalone services. The access rules and other authorisation configurations are sourced from a policy repository.

The UAA solution is based on the well-known NiST ABAC reference architecture.



The UAA solution is based on NiST ABAC reference architecture

5. Runtime Platform

WF

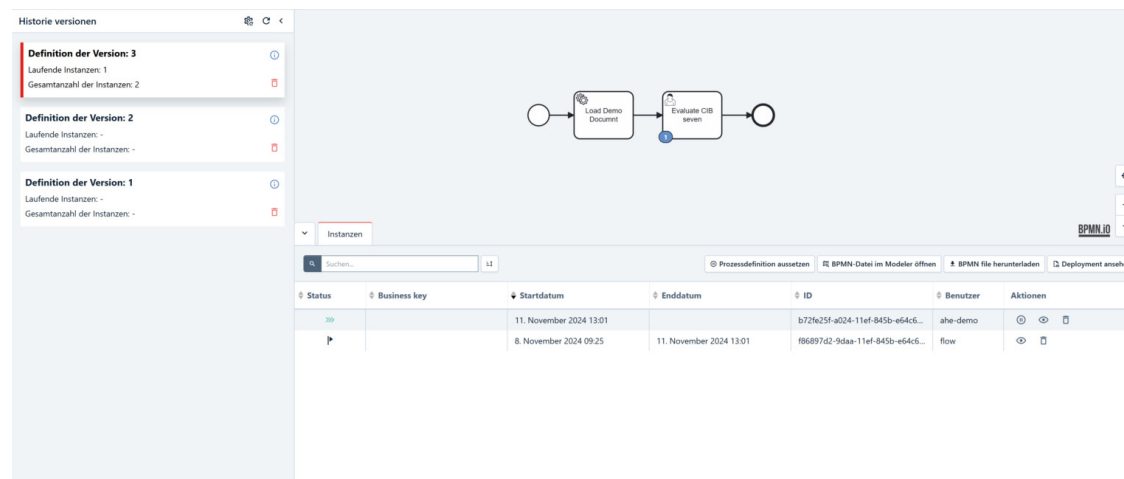
Workflows

A12 Workflows provide a lightweight service that integrates business process model and notation (BPMN) modeling functionality into A12. This makes it possible to perform graphic modeling of server-side workflows and their execution.

The A12 workflow service can be activated as an extension to other A12 products and integrates seamlessly into the A12 architecture.

In this manner, documents can be used as inputs and outputs for A12 workflows, and the user interface for user tasks can be created using the existing A12 modelling approach.

In addition to user tasks, automatically executable tasks such as service tasks or script executions can also be modeled, allowing the realisation of partially and fully automated workflows using process modules. CIB Seven's BPMN Workflow Engine is used as a central component of A12 workflows.



Features:

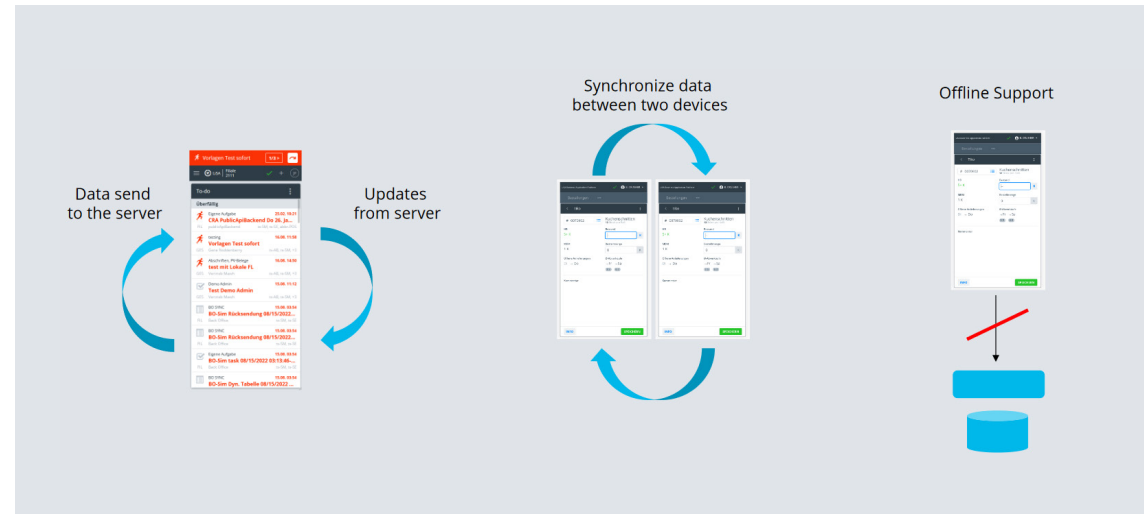
- ⊕ Model driven business processes
- ⊕ Server side / asynchronous / semiautomated
- ⊕ BPMN2
- ⊕ CIB Seven process engine
- ⊕ Camunda model editor
- ⊕ Integrating with A12 modeled data

5. Runtime Platform

DD

Data Distribution

Fast and secure synchronization of data is one of the most demanding technical tasks in full-blown business applications - especially when not all systems involved are permanently online. The Data Distribution component of A12 is a transport layer that specializes in exactly this. The technical service is designed to distribute data between servers and clients and to propagate changes - especially in scenarios where clients are temporarily offline. The component's origins lie in an e-commerce project, in which it manages the data synchronization of a global store network.

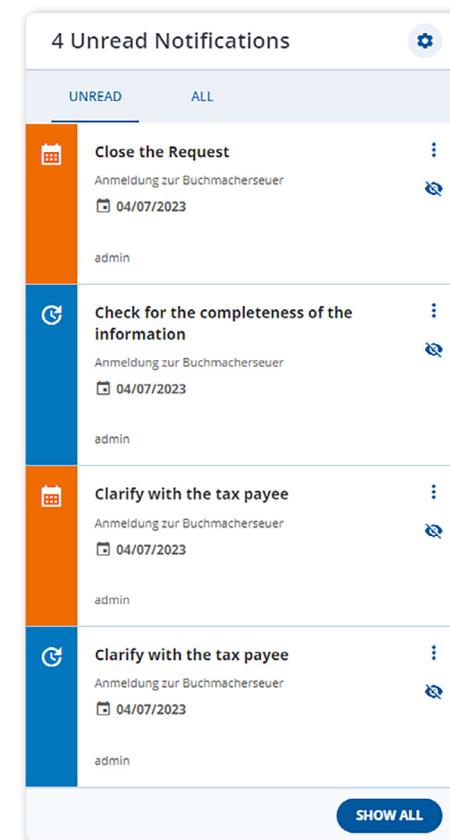
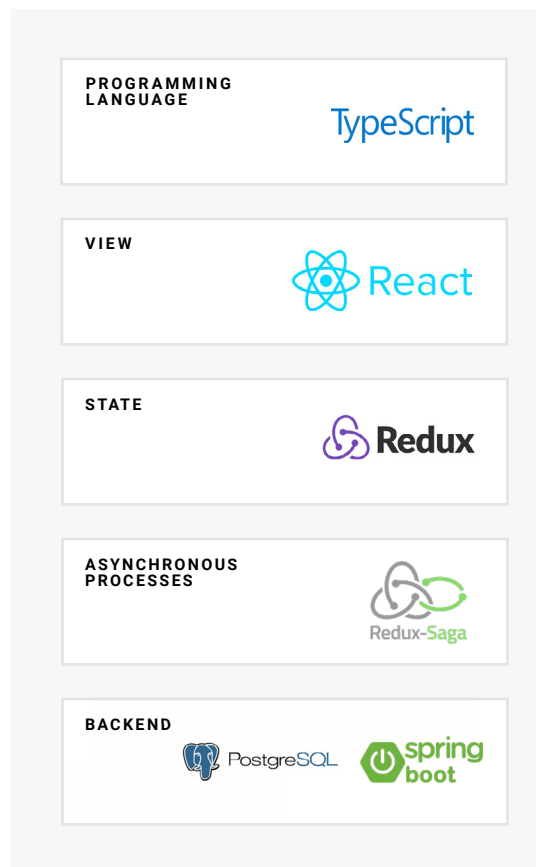


5. Runtime Platform

NC

Notification Center

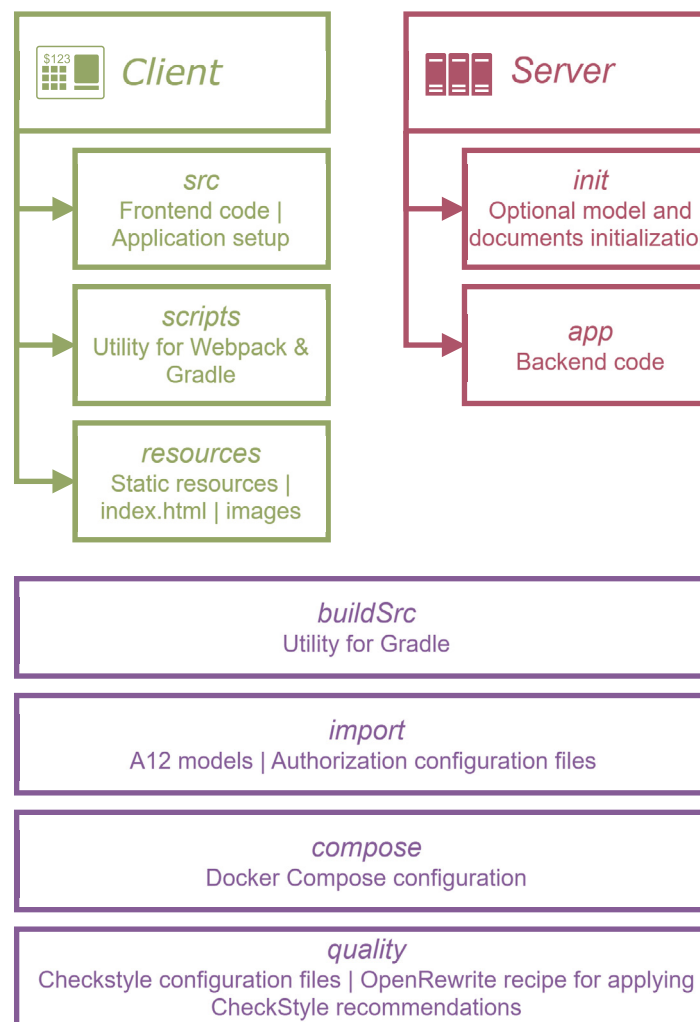
In the context of business applications, employees are typically flooded with a number of different messages. There is information about new tasks, messages from various communication channels, as well as appointments and reminders. With the Notification Center, all these messages in business applications can be bundled in one central location. It serves as **a collection point for different types of notifications** based on different business use cases, structured views, different filters and user preferences. The Notification Center integrates seamlessly with A12-based applications. It provides several predefined notification types. Using the Notification Center's API, the development team can also create their own custom notification types quickly and conveniently.



5. Runtime Platform

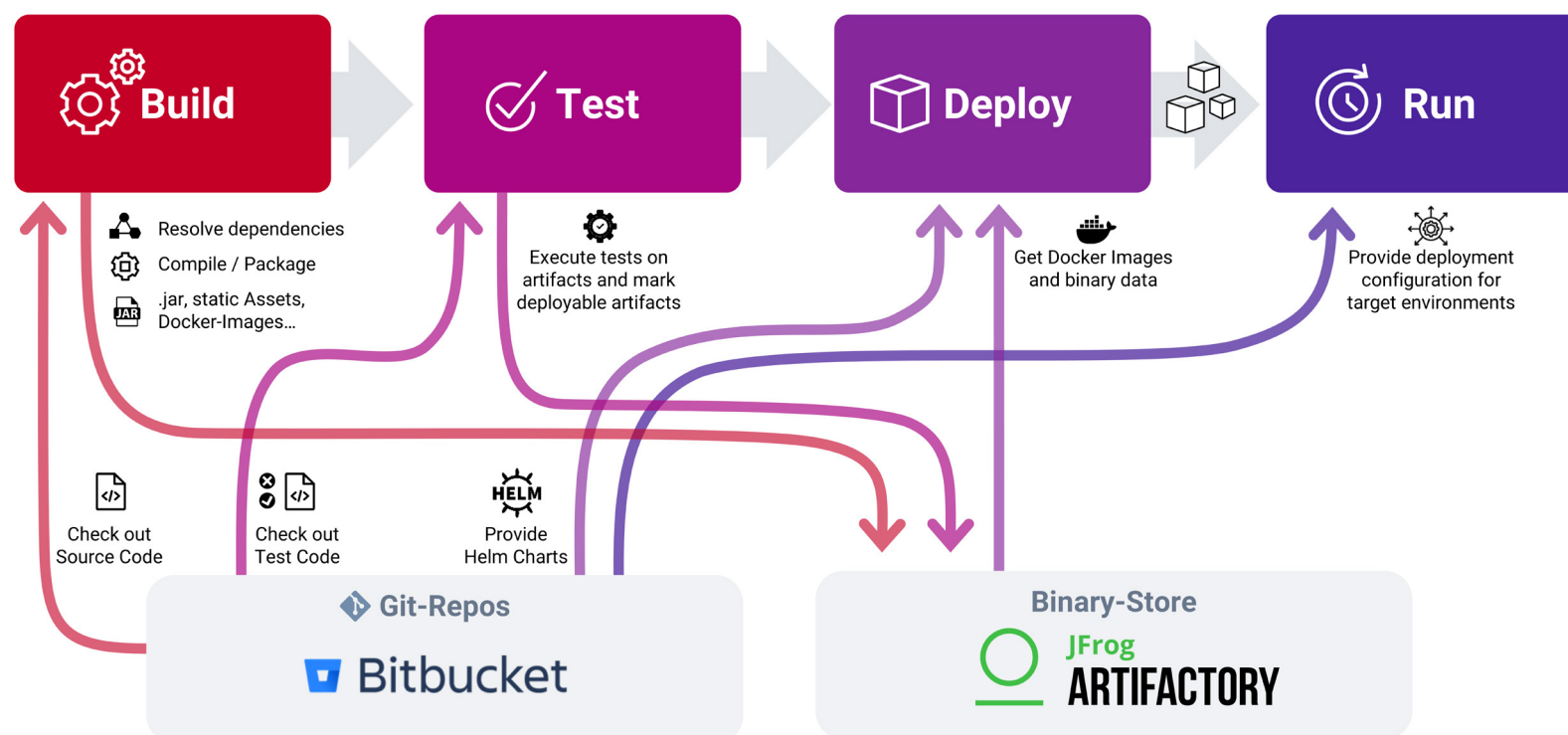
Project Template

The A12 Project Template provides a starting point for development teams to conveniently set up A12 projects and quickly bring A12 applications into production. Among other things, it contains standardized build pipelines as well as development and test environments and covers basic security requirements. At its core, the template includes the A12 components Data Services, Client and UAA. Keycloak is set as the identity provider, and the authentication type is OpenIDConnect/Oauth2 in the default case. Optional components such as workflows, the Notification Center and the Print Engine can be integrated in a standardized way if required.



5. Runtime Platform

Operations



The A12 platform can be run on-premise in the company's own or external data center. In addition, mgm offers hosting in mgm's private cloud in a data center in Germany. Another option is to run it in the cloud with any cloud provider.

Cluster capability - A12 is Kubernetes-ready

A12 applications are designed for deployment on Kubernetes clusters. Built on the A12 project template, A12 provides a standardized way to build and deploy applications to all common development and production environments (DEV, TEST, and PROD clusters).

Operationally, the process involves the following four steps, each of which follows proven standards.

Build – from source code to artifact

The **build phase** is triggered at different times in different stages - e.g. when developers commit and push their code changes to the project's Git repository. In addition, the A12 templates for build and deployment also take into account nightly, feature, and release builds. During the build process, dependencies are first resolved before the code is compiled and packaged. The build pipeline then creates a Docker im-

5. Runtime Platform

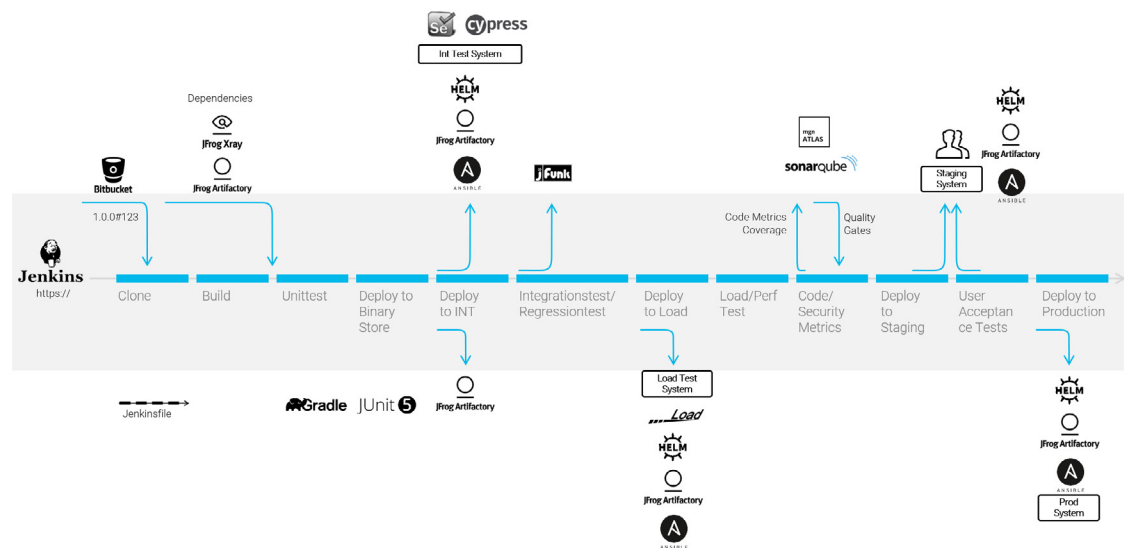
age that encapsulates the application artifact along with its runtime dependencies. The resulting Docker image and all other build outputs – such as JAR files or static assets – are published in the binary store (mgm uses Artifactory by default). Each build is tagged with detailed metadata – including the Git commit hash, build timestamp, version, and build number – which links each artifact in Artifactory directly to the corresponding source code in Git.

Testing – Automated quality assurance

The **testing phase** consists of several steps to ensure that only quality-assured builds progress through the pipeline. Test types include unit tests, integration tests, end-to-end tests, load tests to identify performance degradation from one version to another, and security tests. All test code – from unit tests to integration tests to end-to-end tests – is stored in Git together with the application code, ensuring that the tests remain synchronized with the corresponding application version. The tests are performed using the artifacts created in the build phase and retrieved from Artifactory. This ensures consistency between what is tested and what is ultimately deployed. Only artifacts that pass all mandatory tests are marked as ready for deployment.

Deploy – Configuration and transfer to the target environment

The **deployment phase** essentially consists of two steps: determining the configuration of the application components for specific target environments and transferring the components to the respective target environments. The Helm A12 Stack is available for configuration, enabling out-of-the-box Kubernetes deployment.



In addition to a repository for the program code, each A12 project includes a repository for configuring the environments in which the software is deployed. This keeps the code and configuration neatly separated. Changes to the configuration also automatically trigger specific Jenkins jobs. For example, adjusting the TEST configuration triggers a specific version to be installed in the TEST environment.

At the same time, it is always transparent who deployed which version and when. During deployment, the pipeline retrieves the built and tested Docker images and necessary binary files from Artifactory. Using the configuration data from Git, Jenkins scripts address the respective target environments – such as TEST, STAGE, or PROD – on the corresponding clusters.

Each deployment is carefully logged, ensuring complete traceability. In the event of problems, deployments can be rolled back to earlier artifact versions using Artifactory's versioned storage and Git's comprehensive history.

Run – Reliable operation in the cluster

Finally, the **run phase** comprises the ongoing monitoring and scaling of the applications in production. A12 relies on proven standards for logging and monitoring, among other things. If the respective operating environment does not specify specific logging and monitoring solutions, A12's standardized logging and monitoring setup (based on Loki and Prometheus) can be used. The status of the entire system can be checked at any time via Grafana dashboards.

If urgent corrections or updates are required, new code is committed to Git, which automatically triggers the entire build, test, and deployment pipeline so that updated artifacts are published in Artifactory and redeployed.

5. Runtime Platform

Hosting options for multiple A12 applications

There are a number of options for hosting multiple A12-based applications:

- ⊕ **Isolation via user rights and otherwise “mixed operation”.**
It is possible to run a central A12 platform running multiple separate A12 applications. If, for example, users have access to several specialist applications, the data and model view could be controlled via rights. Services used across all applications can also be shared.
- ⊕ **Isolation through separate deployments**
If the A12 applications are to be run in isolation from each other, the A12-specific services (database, Solr, etc.) must be deployed separately for each separate runtime environment.

Modularization of deployed artifacts

The frontend part of an A12 application can be deployed as an NPM package. The models are deployed separately in the corresponding servers: the workflow model is installed (or updated) in Camunda and the data/form models are injected into the A12 Platform server (via the import API using REST call).

For the communication with surrounding systems several options are possible:

- ⊕ **Make data from the peripheral system available to the client as A12 Documents.**

Option 1: Peripheral system actively pushes data

The data from the peripheral system is actively made available by the surrounding system, e.g. via JMS messaging (transactionally secure) directly to the A12 server (which is extended by JMS listeners for this purpose). Or by calling the data services APIs remotely on the side of the surrounding system. For this purpose we offer a JSON-RPC API with CRUD and other operations. These operations can be sent in batch, which are then processed in a common transaction. But you can also define your own Spring MVC REST endpoint or JSON-RPC “Custom Operations” - this is successfully practiced in many projects.

Option 2: A12 Data Server calls repository on demand (“replaces database”).

One can easily implement a custom Spring repository for a document type that redirects CRUD and list operations to the Umsystem. The repository would then not use JDBC, but work via messaging (JMS) or REST/SOAP. Note here that only JMS messaging runs in Java EE transactions.

- ⊕ **Offer operations of the surrounding system “directly**
If the peripheral system is more likely to offer operations, or the data is to be seen directly by the client (i.e., not as A12-compliant documents), then one can provide a server-side service to the client that serves as a facade/adaptor between the client and the surrounding system.
As a standalone service, this service can be provided via Spring Boot or based on another framework or a non-JVM runtime. Authentication and authorization is provided via A12 UAA. The service can implement the calls internally as desired. REST or better JSON-RPC is recommended as endpoints visible to the client.
- ⊕ **Direct call of the surrounding systems from the client**
This is technically possible. However, direct access to backend systems without UAA is not recommended due to security concerns and SSO/CORS complications.

Appendix A: Technologies

The separation of business expertise and technology allows the technologies used to be exchanged as required. On the following pages you will find an overview of the current A12 technology stack.

6. Appendix A: Technologies

Technologies currently in use

A12 PRODUCT	TECHNOLOGY	DESCRIPTION
K Kernel	Java	
	Typescript	
	Groovy	
	Antlr	Parser generator
	StringTemplates	Template engine
	JAXB	Mapping Java objects to XML
	Jackson	JSON processor for Java
W Widgets	Typescript	
	React	Building UIs
	Styled Components	CSS styling
	Recharts	Chart library
	Lexical	An extensible text editor framework
	React-Dnd	Drag and drop handling
	React-virtualized	Rendering partial data into DOM
UAA UAA	Material Icons	Icon set
	Typescript	
	Redux	State management
	oidc-client-js	OpenIdConnect authentication protocol
	Java	
	Spring	Application framework for the Java platform
	Spring Boot	Auto configuration for Spring application

6. Appendix A: Technologies

		Spring-security	Spring security approach for authorization (SpEL - Spring Expression)
		KeyCloak	Identity and access management
		OAuth2/OpenID	Protocol for authentication
		SAML	Protocol for authentication
		LDAP	Protocol for accessing and maintaining distributed directory information services over an IP network
DS	Data Services	Java	
		Apache Tomcat	Application server
		Eclipse Jetty	Application server
		PostgreSQL	Database
		Spring Security	Authentication, authorization
		Spring Boot	Auto configuration for Spring application
		Typescript API	
WF	Workflows	Kotlin	
		Spring	Application framework for the Java platform
		Spring Boot	Auto configuration for Spring application
		CIB Seven	Platform for BPMN workflow and DMN decision automation
		Typescript	Frontend
		React	Building UIs
		Webpack	JavaScript module bundler
		NPM	Package manager for JavaScript

6. Appendix A: Technologies

O	Overview Engine	Typescript	
		React	Building UIs
		React-Dnd	Drag and drop handling
		React-virtualized	Rendering partial data into DOM
		Redux	State management
F	Form Engine	TypeScript	
		JavaScript	
		NodeJS	Java runtime environment
		NPM	Package manager for JavaScript
		Webpack	JavaScript module bundler
		React	Building UIs
		Redux	State management
		Marked	Markdown in expression language
		Jison	Expression language
		moment.js	JavaScript wrapper for the date object
T	Tree Engine	Typescript	
		React	Building UIs
		React-Dnd	Drag and drop handling
		React-virtualized	Rendering partial data into DOM
		Redux	State management

6. Appendix A: Technologies

C	Client	Typescript	
		JavaScript	
		TSLint	Analysing Typescript
		NodeJS	Java runtime environment
		NPM	Package manager for JavaScript
		Lerna	Managing multi-package repositories
		Webpack	JavaScript module bundler
		React	Building UIs
		Redux	State management
		Inversify	Configuration injection
SME	Simple Model Editor	A12	Frontend
		Typescript	
		React	Building UIs
		Redux	State management
		Redux Saga	Library used to handle side effects in Redux
	Runtime	Docker / Docker-compose	Defining and running multi-container Docker applications
		Kubernetes	Managing containerized workloads and services
		Prometheus	Systems monitoring and alerting toolkit
		Grafana	Analytics & monitoring
		ELK (Elastic, Logstash, Kibana)	Log management
		Ansible	Automating configuration management & application deployment

mgm technology partners GmbH

Taunusstr. 23

80807 Munich

Germany

Tel +49 89 / 35 86 80-0

www.mgm-tp.com

info@mgm-tp.com



Innovation Implemented.