

Enable Business Users to Embed Dynamic Database Content in Existing Web-Based Systems Using Web Components and Generic Web Services

Andreas Schmidt^{1,2} ^a and Tobias Münch^{3,4} ^b

¹*Institute for Automation and Applied Computer Science, Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany*

²*Department of Computer Science and Business Information Systems, Karlsruhe University of Applied Sciences, Karlsruhe, Germany*

³*Münch Ges. für IT Solutions mbH, Gewerbering 1, 49393 Lohne, Germany*

⁴*Chemnitz University of Technology, Chemnitz, Germany*

Keywords: Web Component, Relational Database, Low Code, Prototyping.

Abstract: In our digitalized world and under the economic pressure of competition, every company must react flexibly to opportunities and problems that arise. One way to cope with these challenges is to use web-based Enterprise Resource Planning (ERP) or Customer Relationship Management (CRM) Systems, which provide significant functionality inside their system range. Third-party systems often have to be integrated with ERP or CRM systems but cannot be connected, for instance, because of limited Application Programming Interfaces (API) or data structures. Therefore, such tasks are complex and time-consuming and must be done by software engineers, who are limited resources in today's enterprise context. However, HTML documents can be integrated with web-based systems such as ERP or CRM, and HTML creation is not limited to the software engineering workforce. Our low-code environment, which is based on W3C web components standards and RESTful web services with state-of-the-art authentication approaches, could solve the shortage because we empower business developers to embed dynamic database content declaratively in static HTML pages or web-based systems such as WordPress or SoftEngine ERP-Suite. Our system also allows the declarative integration of forms for creating/modifying and deleting data records (CRUD functionality). The low-code web components access the database via the RESTful service. The API of the RESTful service abstracts the database manufacturer-specific characteristics, such as the storage format of the metadata.

1 INTRODUCTION


Today, first-world countries are grappling with a shortage of skilled software engineering workers, a challenge exacerbated by the competitive nature of our businesses and the increasing importance of digitalization (Breux and Moritz, 2021; Hyrynsalmi et al., 2021). However, there is a beacon of hope in the form of low-code solutions, which offer a promising way to address this pressing issue (Elshan et al., 2023).


In Germany, companies are categorized as micro, small, middle, and enterprise companies (StB, 2024). Micro and small companies often have limited budgets and liquidity for information technology expenses (Almaree et al., 2015). Therefore,

they usually use several cloud applications as Software as a Service (SaaS) and have an external IT service provider (Bajenaru, 2010). In comparison, mid-sized and larger companies often have IT departments, which are frequently overloaded with administrative and maintenance tasks (Grooss, 2024).

So, the main challenge is that a company's users often have different business challenges that must be handled quickly, but the IT resources of companies can not cope with these load peaks. However, the average business developer should have achieved basic HTML knowledge in school, as the Committee on European Computing Education (CECE) reported (CECE, 2017). What if these users could handle most minor challenges using a low-code HTML library?

Our proposal is a versatile low-code library of web components connected to our generic RESTful (Richardson et al., 2007) web service, designed to seamlessly integrate third-party databases into var-

^a  <https://orcid.org/0000-0002-9911-5881>

^b  <https://orcid.org/0000-0001-9424-6201>

ious web-based systems. Whether it's an ERP system, a CMS system like WordPress, a SaaS solution, or even a simple HTML file in an arbitrary directory on a local PC, our solution is adaptable to the unique needs and IT setups of companies of all sizes.

The rest of the paper is structured as follows. In Chapter 2, we present the scenarios for our proposal and how it would affect the main stakeholders. Afterwards, we describe in Section 3 our work regarding existing research, such as database access through the web, web standards, related solutions and low-code approaches. Upon the existing research, we describe our overall system architecture in Section 4. Then, we focus on the detailed implementation of various web components and their purpose in Section 5. Afterwards, the usage of our system is described in Section 6 with an example application in a real-world scenario for a distributor of food and beverages. Before we summarize our current state of research in Section 8 and point out future work, we will discuss current limitations and possibilities for improvement in Section 7.

2 SCENARIOS

In theory, enterprise companies have various systems to store, process, and provide information securely and safely for several departments to provide business value for their customers in a unified IT environment (Minoli, 2008). The cross-cutting IT department monitors, maintains, and extends the enterprise IT architecture for the whole company cost-efficiently (Minoli, 2008). However, the enterprise software architecture in mid-sized companies is more heterogeneous, with various applications and data structures for a particular department, such as a schedule for human resources interviews. This discrepancy can be explained by Conway's law, which states that every company will produce a copy of its own organizational structure inside a developed system (Herbsleb and Grinter, 1999).

However, the IT Team in mid-sized companies often finds itself in a challenging position. Despite their best efforts, they struggle to fulfil special integration requests from internal clients (Haug et al., 2011). This is primarily due to their limited resources, which are mainly dedicated to maintaining the central systems. As a result, they cannot cater to the diverse needs of the various departments, leading to a significant gap in service delivery.

These distinct applications for specific use cases often must change or be integrated quickly and resiliently to fit changing market requirements. There-

fore, a low-code approach could be an enabler for digital transformation in the industry because users with limited programming knowledge could handle many tasks themselves (Prinz et al., 2021). According to the CECE's work, citizens probably used HTML in school because they should be fluent with standard IT tools such as web browsers (CECE, 2017). Additionally, web-based business applications' market share is increasing, and in these applications, HTML documents could be used to extend the functionality (Business Research, 2024).

Our proposed approach does not completely overhaul the existing systems. It acknowledges the users' background and the presence of existing systems. It assumes that some of the company's applications are based on databases that can be connected to our generic RESTful web services. They provide the data and meta information, such as attribute details and the actual dataset, making them a valuable resource in our procedure.

The following sections explain different scenarios based on the perspectives of administrators, business developers, and users to define the specific working environment for our low-code HTML technique.

2.1 System Administrator

The System Administrator is the backbone of each company's IT department. They are responsible for ensuring the smooth operation and maintenance of the current infrastructure, including software and on-premise hardware. Typically, IT workers have completed an apprenticeship (44%) or used a lateral entry (23%) and sometimes a computer science study (16%), underscoring their expertise and dedication to their role (Statista, 2024).

In our case, he should accomplish the new task of installing, configuring and connecting the RESTful Webservice to one or more internal databases such as Postgres or MySQL. The databases are linked through corresponding entries in a configuration file where the access information is stored and the authentication mechanism is specified.

In companies, it is necessary to recognize a role-based access and control system (RBAC) such as Keycloak or an Active Directory so that only a distinct group of users can access a specific data source. For example, this could be realized with a role through Keycloak, which is then provided inside the Java Web Token (JWT). The RESTful Webservice examines this JWT.

2.2 Business Developer

With their expertise in building business processes, the business developer is a critical player in the system. The stakeholder is part of Business Development, which Uittenbogaard et al. describe as 'involves the actual development of product-market combinations; in other words, it consists of the execution of the innovation process' (Uittenbogaard et al., 2005).

In our case, they are the drivers of business improvements, and their role is crucial in ensuring that the system meets the needs of the users and the company's business processes. They integrate the low-code web components into web-based systems and connect them to the previously connected databases using the configuration specified by the system administrators.

2.3 User

Finally, users, such as service agents, interact with the integrated web components and use them and their interaction options as part of the business process. Several examples of how a user would use the provided web components include seeing data on a central monitor, additional information for order through a parts list provided by a construction software system, or further appointments connected with a procedure. In Section 6, we will focus on a use-case inside the purchasing department of a distributor for food and beverages.

3 RELATED WORK

According to the scenarios described in the previous section, our focus lies on the related works of database access through the web, Java applets, and low-code development because each approach integrates different aspects of our work.

3.1 Database Access Through the Web

The classic approach to bringing database content to the web are server-side web applications designed and implemented for a particular field of application (Florescu et al., 1998). Specialized frameworks such as Django (Python), Symfony (PHP), or Express (JavaScript) are also often used for this purpose. Most of these implement a form of the model-view-controller (MVC) paradigm and use an object-relational mapping framework to access the database.

On the one hand, this requires knowledge of the programming language or the framework used, as well as access to the corresponding development and runtime environment. On the other hand, the present work takes a client-side, declarative low-code approach, in which database content is integrated declaratively into web pages. The W3C composite standard *web components* (Web, 2024) is used for this purpose.

3.2 Web Components

Web components are a W3C standard supported by all recent web browsers and allow the definition of custom HTML elements (Web, 2024). For this purpose, a developer creates a JavaScript class of custom web components that defines the new HTML element's behaviour and the browser's visual representation. The newly defined element can be used once the JavaScript class is included in an HTML page. As the entire logic is encapsulated within the JavaScript class, no other runtime environment besides the browser is required, such as a server-side application server. Of course, the components can communicate with external services if needed, but in principle, no further services are necessary to run the web components.

The JavaScript class that defines the logic and visual appearance of the new element must inherit from the `HTMLElement` class (Web, 2024) and, in addition to the actual application logic, implement a series of predefined methods that realize the integration of the element into the Document Object Model (DOM) tree of the HTML page.

Custom HTML elements have a naming convention that specifies that the names must have a hyphen (such as `<db-table>`) and that they must have an opening and mandatory closing tag in the HTML page (`<db-table name="city" ...></db-table>`).

3.3 Java Applets

Java applets (Boese, 2009) follow a similar approach to web components. Both approaches run on the client within the browser, and some predefined methods must be implemented to integrate the application into the HTML page. The integration is declarative in both cases and parameters can be passed to the application. The programming model allows programmatic access to the DOM tree of the embedding website, similar to the web components. With Java Database Connectivity (JDBC) (Reese and McLaughlin, 2003) there is also a sophisticated access API for relational databases available. The applets are integrated into the web pages using the elements `embed`

or object. From the mid-2010s, however, browser manufacturers gradually discontinued support for applets (Oracle, 2020).

3.4 Low Code Development

The evolution of low-code development platforms will significantly transform software engineering tasks and help business developers build applications themselves without being too deep into coding (Prinz et al., 2021). Low-code development platforms (LCDPs) offer a visual approach to application development, primarily through model-driven design and declarative programming (Elshan et al., 2023; Prinz et al., 2021). The LCDPs also provide web applications as an output, such as the low-code platform Xelence by Sagitec Software (Arora et al., 2020).

In the study of Prinz et al. the participants mentioned using low-code solutions because of the reduced coding effort and the low learning curve (Prinz et al., 2021)

As part of the low-code initiative, a number of application builders have emerged, such as Caspio (Caspio, 2024), Budibase (Budibase, 2024), webflow (Webflow, 2024), and Bubble.io (Bubble, 2024), which make it possible to develop web-based applications in the form of single-page applications with little or no programming effort. However, these represent a self-contained system that cannot be integrated into existing web applications, or only with difficulty. Our low-code web components provide such functionality with ease of development, interoperability, extensibility and maintainability.

4 OVERALL ARCHITECTURE

Figure 1 illustrates the overall architecture of the application. The developed web components (with prefix *db-*) operate in the browser and communicate with each other through method calls or via a public-subscribe mechanism.

Communication with the databases does not occur directly between the components and the database but between the component *db-server* and a RESTful web service (*rdcms.php*), which acts as an interface to the connected relational databases. We used the *php-crud-api* REST Service (van der Schee, 2024) by Maurits van der Schee as a core, which we extended with a number of additional modules. In addition to *php-crud-api*, there are a number of other existing libraries, such as *db2rest* (db2rest, 2024), which could have been used instead. For an overview, see (Bohdan, 2024). The web service provides the

following functionality:

Database Selection: To support multiple databases dynamically, a separate entry point (*rdcms.php*) was implemented. Databases that should be accessible from the RESTful API are specified in a configuration file. The authentication method is also defined at this point. Each database is specified by a logical name under which it can then be accessed from the web components. Figure 4 (middle) shows the structure of the entries in the configuration file.

Access Control: The *php-crud-api-server* already offers an interface to use different authentication mechanisms like authentication via API key, username/password combination and JWT token (Peyrott, 2024). All these mechanisms are based on transporting authentication information in the HTTP header. To also support local HTML files accessed via the file URI schema (Kerwin, 2017), we have integrated an additional mechanism that sends an API key as an HTTP parameter. The reason for this are CORS (Hossain, 2014) restrictions that would otherwise not allow access from the web components to the web service when accessing the HTML document via a file URI (like `file:///tmp/dbwc-demo.html`). This type of access places the least demands on a business developer, as only an editor to create the HTML documents and a browser with an internet connection are required to access and visualize the database content.

CRUD Functionality: For retrieving the data for a single table and the modification of datasets, we use the functionality of the previously mentioned RESTful *php-crud-api* service. The software is very easy to install, by only copying a single PHP script to our web server and provides the full functionality we need to extract, create, modify, delete, filter and sort datasets. The software can also be easily extended by defining and integrating your own controllers.

Metadata Access: This extension returns metadata about the available databases. These include (1) available databases, (2) available tables per database, (3) columns, data types and constraints of the individual tables. The constraints include primary keys, foreign key relationships, not null, and unique constraints.

Metadata plays an important role, as it allows, for example, the resolution of foreign key relationships, the monitoring of non-null constraints, and the performance of data type checks on the client side. Furthermore, visualization and editing op-

tions corresponding to the data types can be implemented.

SQL Module: The SQL module extension makes it possible to directly issue SQL statements to the relational database. The statements are passed as arguments from the `db-query` web component (Section 5.5), which is then also responsible for displaying the result data records.

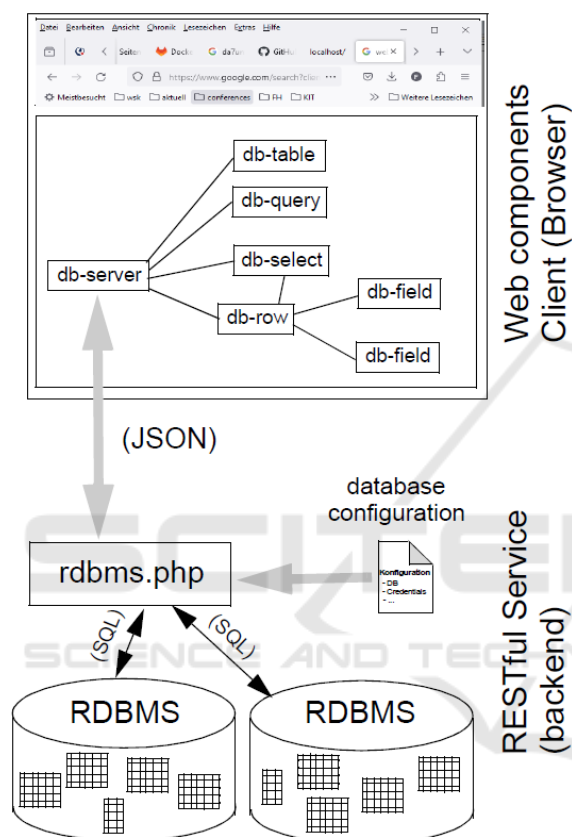


Figure 1: Architecture of our database web components. The module `rdbms.php` receives the requests from the `db-server` web-component and forwards them to the database specified in the request. The access information for the database is stored in the database configuration (see Figure 2) on the server. The results are then returned to the web components via the `rdbms` module.

5 COMPONENTS

As part of our research work, we have developed a series of web components for the declarative integration of database functionality into HTML pages.

db-server: This component establishes the connection between the components and the backend. Except for user authentication, it has no visual

representation, but is used by the other components to communicate with the database via the RESTful service.

db-table: Component that represents a database table or a part of it.

db-row: Component representing a single dataset (row in a table). The functionality of this component ranges from the simple predefined visualization of the component to the creation and editing of datasets using internal or external forms to its use as a controller in a Model-View-Controller setup.

db-field: Component that represents a single attribute of a `db-row` component.

db-select: The functionality corresponds to the HTML select element, in which an entry can be selected from a series of predefined values. In the database context, it can resolve and set foreign key values.

db-query: Component that represents an SQL select query.

In the following section, we will present these components in detail. The data in the example screenshots shown comes from the *Mondial* database (May, 1999). Since the `php-crud-api` module we use does not support updates of records with composite keys, we have modified the schema of the *Mondial* database by replacing the composite keys with artificial numerical keys and adjusted the foreign keys accordingly.

5.1 Server-Component

The first component we present is responsible for communicating with the web service. The server component can occur several times so that multiple databases can be accessed from one HTML page.

Figure 2 shows the mapping between the `db-server` component inside a HTML page to a specific database. The component `db-server` specifies the endpoint of the RESTful service and a logical database name using the parameters `url` and `database` (left side of the figure). On the REST-API side (middle), there exists a configuration file that maps the logical database name (i.e. `webist-demo`) to a specific database (right side of the figure). This mechanism is similar to ODBC or Oracle `tns-names` (Oracle, 2024). Note that the database can run on any computer and not necessarily on the computer with the REST-API endpoint.

To access the URL of the RESTful service from the browser, you must either allow Cross-Origin Resource Sharing (CORS) on the server or configure a

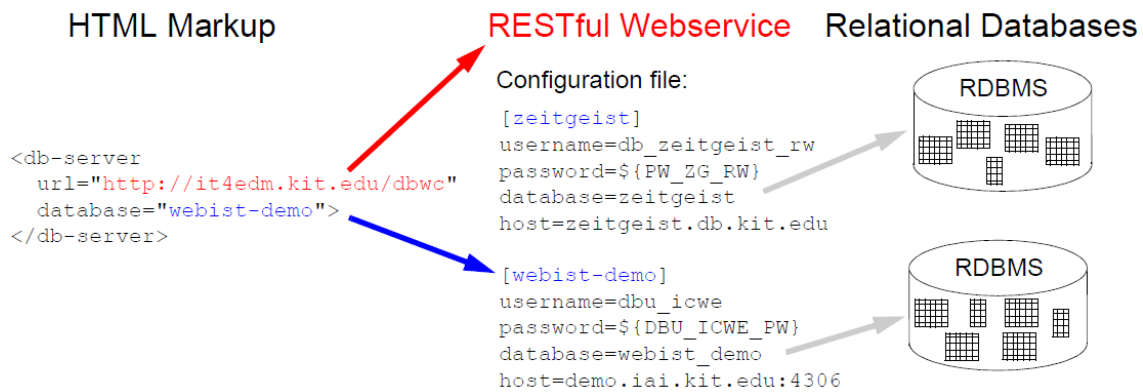


Figure 2: Database mapping - the RESTful service maps the logical database name (*webist-demo*) specified in the web component (left-side) to the access information stored in the database configuration file (middle) and uses this to access the specified database (right side).

reverse proxy that forwards the request from the original host to the specified URL, if the URL points to another host.

5.2 Table-Component

The *db-table* component is responsible for displaying the content of a database table or a part of it. The functionality ranges from the static display of a table with pre-defined conditions (selection) on the data records and the selection of certain columns (projection), to a highly interactive component that reacts dynamically to changes in the table content and offers a range of interaction options.

Figure 3 shows in the upper part the appearance of the component *db-table* inside the browser. The component allows page-by-page scrolling (1), sorting by column values (2), in-line editing (3), and the formulation of additional conditions (4) as well as the possibility to edit/delete datasets (5).

The corresponding HTML-code is shown in the lower part of Figure 3. Beside the *db-table* component, also the corresponding *db-server* component is shown.

The *attribute-list* attribute defines the columns to be displayed, while the *action* attribute defines the possible functionalities of the component. Specifically, *inline-edit* (3), *filtering* (4), as well as editing and deleting data records (5) and *pagesize* attribute specifies the number of data records per page. In addition to the parameters from Table 1, the component's appearance can also be adapted to your requirements using cascading stylesheets (CSS).

Name	Islands	Area
Bahamas	Bahama Islands	13935
Corse		8681
Cuba	Grosse Antillen	114524
East Falk	X S Falkland Islands	12173
Fidschi	Fidschi	18272
Fuennen		2976
Gheschm		1491
Gotland		3140
Greenland		2175600
Hispaniola	Grosse Antillen	76192

```

<db-server url="https://db.kit.edu.de/dbwc"
  database="mondial">
</db-server>

<db-table table="island"
  attribute-list="Name, Islands, Area"
  actions="filter, edit, delete, inline-edit"
  pagesize="10">
</db-table>
    
```

Figure 3: Visual appearance of the web component *db-table* (top) and the corresponding HTML code (bottom). The attributes *attribute-list*, *action*, and *pagesize* specify the visual appearance and behaviour of the table.

5.3 Row-Component

The component *db-row* has several tasks. The standard configuration allows you to edit a data set specified by the primary key in a formula provided by the component. This component extensively uses the metadata provided for a table by metadata module of the RESTful-service. This applies to information about the data types of the fields, the not null constraint and information about foreign-key relationships.

This function is shown above in Figure 4, which displays the dataset for "France". Here, the value of the foreign key attribute *Capital*, which refers to the capital of a country, is not displayed as a numerical foreign key value, but a selection field that displays the referenced value and simply offers the possibil-

Table 1: *db-table* Attributes.

Attribute	Description	Mandatory
table	Name of the table	yes
filter	Mandatory filter condition, every dataset must fulfil	no
pagesize	Maximum number of rows on a page	no
order	Sort order (column name)	no
direction	Sort direction (<i>asc</i> , <i>desc</i>)	no
page	Page to display	no
connection	Id of a <i>db-server</i> web component. If the attribute is not set, the default server component is chosen	no
attribute-list	Comma separated list of attributes to display (default: all)	no
actions	Allows the activation of "inline-edit", "delete", "edit", "filter", and "paging" functionality	no
refresh-rate	Time in seconds after which the table data is reloaded from the database	no

ity to change it using the full-text search function of the web component *db-select* described later in section 5.4.

The lower part of Figure 4 shows the corresponding declaration of the *db-row* element inside the HTML page. The *key* attribute is used here to provide the primary key value of the dataset of "France". If the attribute *key* is omitted, the component offers a form for creating a new data record.

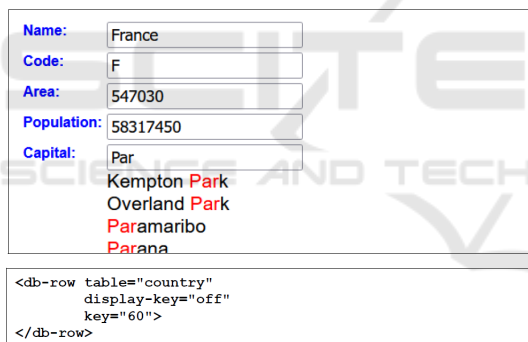


Figure 4: Web component *db-row*: Visual presentation (top) and embedding in a HTML page (bottom). The record with the primary key value 60 (attribute *key*) of table *country* (attribute *table*) is displayed. The foreign key column *Capital* is resolved and displayed as a *db-select* component in which the possible values can be selected.

The component has various options for customizing the appearance of the provided form using CSS. If these design options are not sufficient, the component also offers the option of working with external forms. This leads to significantly more possibilities for influencing the layout.

If the dataset should only be displayed but not changed, there is a "read-only" mode in which the dataset is shown in tabular form (2 columns). Again, style sheets can be used to customize the layout. The additional use of the *db-field* component is even more flexible in terms of visualisation. In this case,

several *db-fields* are linked to a *db-row* component. The *db-row* component then only serves as a data source for the *db-fields* components but is no longer responsible for the output, which is then taken over by the *db-fields* components. This allows any desired page layout.

Using the *controller* parameter, the component can be used as a controller within an Model View Controller (MVC) setup. In this case, the component reads the values of a dataset either from the GET parameters of the current URL or from a JSON string passed to the *data* parameter and writes the dataset to the database. Depending on the transferred attributes, an SQL INSERT or an UPDATE statement is executed. The distinction is made on the basis of the existence and value of the primary key attribute. Alternatively, another attribute can be specified that takes over the role of the primary key. If a data record already exists in the database with a matching value for this attribute, an SQL-UPDATE is performed. Otherwise, an INSERT operation is executed. After executing the SQL statement, the web component loads the referring page or an arbitrary page, which can be specified with the additional parameters *target-url* and *error-url*. In Section 6 an application in the context of the MVC paradigm is shown, in which a page containing a form acts as a view and the HTML page with the *db-row* component takes over the controller part.

5.4 Selection-Component

The *db-select* web component shows a selection box from which values can be selected. The values of the selection box can either originate from a table by specifying the table name and an optional condition or be specified through an SQL statement. Analogous to the behaviour of the standard HTML-select box, one or two values can be specified. The first value is the

value to be returned from the form, and the second is the text to be displayed. A simple text-match search and a prefix search are offered to find the entries you are looking for. Figure 5 (top) shows the appearance of the *db-selection* box. The search text entered is highlighted in colour in the matching entries. The underlying markup code can be seen in the lower part of the figure. Attribute `table` specifies the table where the values come from. The attribute `name` represents the name of the element within the formular (element form), which is passed on to the formular together with the selected value. The attribute `label` names the column in the table, from which the values being displayed are taken, while the attribute `key` specifies the column whose value is passed to the form.

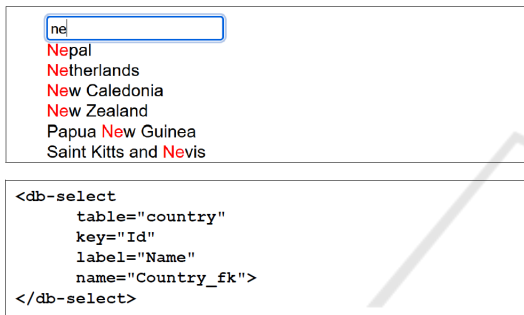


Figure 5: Web component *db-select*. Prefix search in column name (attribute `label`) of table `country` (attribute `table`).

5.5 Query-Component

The *db-query* component is quite similar to the *db-table* component. The main difference is that no table and optional parameters such as `filter` and `order` are specified, but an arbitrary SQL select statement. The upper part of Figure 6 shows the component's appearance in a browser, while the lower part shows the embedding of the web component in the HTML page. The provided SQL-Statement, returns the number of bordering states and the total length of external borders for each country, sorted by total border length in descending order.

Just as with the *db-table* component, additional optional parameters such as `refresh-rate`, `pagesize`, etc. can be specified.

5.6 Schema Independence

Due to the consistently generic structure of the components and the RESTful service, our components are independent of any schema changes to the database (as long as the tables used continue to exist and no columns explicitly specified in the components are

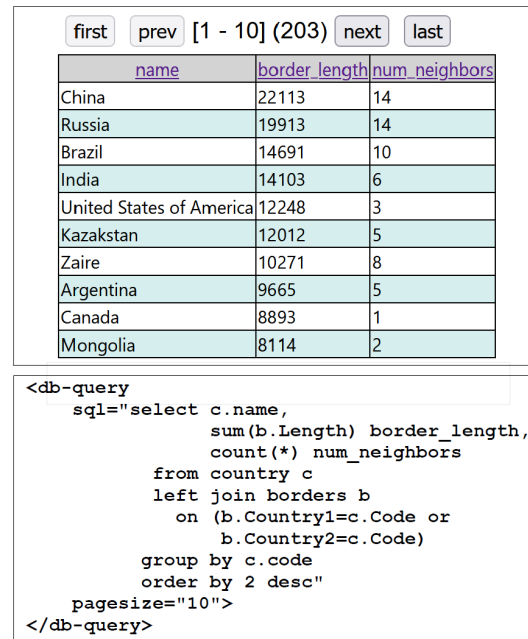


Figure 6: Visual appearance of the web component *db-query* (top) and the corresponding HTML code (bottom).

omitted). This is realized by the components and the RESTful service having access to the meta information of the relational database, which always returns the current state of the database schema and is the only basis for structural information within the components.

5.7 Customization

The components are displayed with a standard layout. However, this layout can be customized using CSS and a series of possible attributes for the components. Additionally, the *db-row* component can be adapted to the layout preferences using an external HTML form or additional *db-field* components.

As an example, Table 1 gives an overview of the supported attributes for the *db-table* component.

6 EXAMPLE APPLICATION

Our example scenario involves a distributor of food and beverages for gastronomy operating within a web-based ERP system that also includes a Customer Relationship Management (CRM) module. In this organisation, the purchasing department has a second application that lacks API connectivity to the CRM. Therefore, users must email the purchasing department if they request new items to purchase. Introducing our low-code approach into this environment

View webist-demo.html (Screenshot)

The screenshot shows a web application interface. On the left, there is a form for 'Allgemeine Daten' (General Data) with fields for 'Adressnummer' (24098), 'Firmenname' (Echtstart GmbH), 'Straße' (Gewerbering), 'Hausnummer' (1), 'Postleitzahl' (49393), 'Ort' (Lohne), 'Land' (DE : Deutschland), and 'UST-ID-Nummer'. On the right, there is a section for 'Artikelwünsche / order wishes' with input fields for 'Article:', 'Priority:' (choose priority), 'Deadline:' (tt.mm.jjjj), 'Amount:', and 'Unit:', along with an 'Add Wish' button. Below this is a table with 10 rows of article data.

id	article	priority	createdAt	deadline	amount	done	unit
1	Milk	high	2020-12-01	2020-12-02	2.50		pcs
2	Bread	medium	2020-12-01	2020-12-02	1.50		pcs
3	Butter	low	2020-12-01	2020-12-02	1.00		pcs
4	Cheese	low	2020-12-01	2020-12-02	0.50		pcs
5	Eggs	medium	2020-12-01	2020-12-02	6.00		pcs
6	Sugar	high	2020-12-01	2020-12-02	1.00		pcs
7	Salt	low	2020-12-01	2020-12-02	0.50		pcs
8	Pepper	low	2020-12-01	2020-12-02	0.50		pcs
9	Flour	medium	2020-12-01	2020-12-02	1.00		pcs
10	Rice	low	2020-12-01	2020-12-02	1.00		pcs

Controller webist-demo-controller.html (HTML Markup)

```

<script type="module" src="http://www.smiffy.de/dbwc/db.js"></script>
<db-server database="webist-demo">
</db-server>
<db-row controller="true"
      table="wishlist"
      action="store-from-get-request" >
</db-row>
    
```

Figure 7: Example Application: Embedded web components in customer view of the SoftEngine ERP-Suite (View with HTML form and db-table component (top, right), controller markup (bottom)).

will streamline the business process and significantly support our stakeholders.

Internal and external service agents are responsible for their customers. During phone calls or on site, the customer confronts the service with requests for new articles, when they need them and how much they would like to purchase per week. The service agent can enter the demand information on the customer data sheet through the low-code web components for the purchasing department embedded in the CRM. In Figure 7 (top), our web component table is included in the SoftEngine ERP-Suite (Eggert and Meier, 2010). The data directly persisted in the purchasing system, and the purchaser could operate with the data. The service agent can see if the customer demand was accepted by the attribute `done` through a status dashboard provided by the web components inside the CRM.

So far, we have seen how the service agents interact with the integrated web components in this business process. However, the system administrator and business developer must first integrate these components. The system administrator's role in integrating the web components is crucial. He makes the relevant tables in the purchasing system database accessible through a new database user for the web components. Then, he defines the connection settings within the configuration of the data-providing web service, which they had installed previously. The system administrator then hands over the name of the database configuration file to the business developer.

The business developer integrates our low-code

web components. In this specific case, it uses a `db-server` component for accessing the database, a `db-row` component that enables the creation of a new customer request, and a `db-table` component that lists the previous customer requests. How the integration looks is described in Section 5 and will therefore not be discussed further here. The business developer uses the system administrator's configuration to connect to the database via the RESTful web service. He is also responsible for matching the user requirements, such as selecting the right attributes and features, to build a solution that meets the needs of the target user group. In our case, the users are our distributor's service agents. In the lower part of Figure 7 you can see the complete markup code of the controller component. The `db-row` component runs in controller mode (`controller="true"`) and writes the key-value pairs of the GET request to the `wishlist` table (`action="store-from-get-request"`). With the `db-server` component it is noticeable that the parameter `url`, which specifies the RESTful service endpoint, is not given. In this case the component assumes that the `url` matches the path of the `src` parameter from which the web components were loaded (`<script src="...">`).

7 LIMITATIONS AND IMPROVEMENTS

In this section, we describe the technical limitations of our web components and the room for further im-

provements. These limitations are structured in (1) look and feel, (2) parameterization and configuration and (3) security and network.

In Figure 7, we see the database column names, which are not always intuitive. To overcome this issue, we must dynamically integrate display names for each column. In addition, the resolution of foreign keys is currently not yet supported by the component `db-table`. This should be optional in future versions. However, this can already be simulated by the alternative use of the `db-query` component and corresponding SQL-join operations. Another point is the extension of the `db-select` web component to enable the selection of multiple entries. This would allow the manipulation of $n : m$ relationships between data records.

Support for parameterizable SQL statements would also be interesting. For example, this would allow displaying additional information from other tables for a specific data set without using JavaScript functionality. In this case, the parameters used would refer to a `db-row` or `db-field` component (i.e. a foreign key attribute). For the future, we are also planning a wizard that will make it possible to interactively build the desired HTML tags with the necessary parameters to be integrated into the HTML pages.

The authentication and authorisation mechanisms are currently limited because we do not support mechanisms like OAuth2 or Open ID connect. The Microsoft Active Directory connection would be beneficial in an enterprise context. In addition, we need to implement a detailed RBAC system so that users only see specified columns or edit the dataset only if they have a particular role. Regarding network access, we have no fallbacks if the web services are unreachable by the client's device. This could happen if the user operates outside the company and the web services are only reachable in the local area network segment.

8 CONCLUSION AND OUTLOOK

In this paper, we have developed and prototypically implemented a concept for a client-side, declarative integration of dynamic database content within HTML pages as a low-code technique. To this end, a series of components for interaction with relational databases were developed using web components technology without needing an individual server-side scripting language or web framework. On the server side, we have implemented a RESTful API service as an interface between the components and the connected databases.

The application spectrum of the components

ranges from the rapid development of a functional prototype that uses the internal capabilities for creating, editing and displaying data records to pages that use the `db-row` component in conjunction with the `db-field` components or external forms can therefore implement any layout.

Users of content management systems such as WordPress can also use the components, as only HTML pages need to be adapted to integrate database content. Beside the RESTful service, which is typically provided by the IT department, no web server is required, as the components also work with the browser's file URI schema.

Building on this basic functionality, we have identified future enhancements such as usability, detailed security with RBAC and configurability. These enhancements are influenced by the limitations of our low-code web components described in the previous Section 7.

Our current prototype can help solve the challenges of our described stakeholders, like the shortage of IT personnel, and empower business developers. Still, we have to evaluate it with the described stakeholders in a company environment to evaluate the developer experience of our low-code package.

REFERENCES

- Almaree, K., Bowman, A., Berenice, B., Visser, C., Bergoer, D., Fullard, D., Moses, G., Brown, S.-L., Bornman, J., and Bruwer, J.-P. (2015). The usefulness of cash budgets in micro, very small and small retail enterprises operating in the cape metropolis. *Expert Journal of Business and Management*, 3(1).
- Arora, R., Ghosh, N., and Mondal, T. (2020). Sagitech software studio (s3)-a low code application development platform. In *2020 International Conference on Industry 4.0 Technology (I4Tech)*, pages 13–17. IEEE.
- Bajenaru, A. (2010). Software-as-a-service and cloud computing, a solution for small and medium-sized companies. *Bulletin Of The Transilvania University Of Brasov. Series V: Economic Sciences*, pages 173–184.
- Boese, E. (2009). *An Introduction to Programming with Java Applets*. Jones & Bartlett Learning.
- Bohdan, D. (2024). Github: `dbohdan/automatic-api`. <https://github.com/dbohdan/automatic-api>. (Accessed on 2024-09-16).
- Breaux, T. and Moritz, J. (2021). The 2021 software developer shortage is coming. *Communications of the ACM*, 64(7):39–41.
- Bubble (2024). Bubble: The full-stack no-code app builder. <https://bubble.io/>. (Accessed on 2024-09-16).
- Budibase (2024). Github: `Budibase/budibase`. <https://github.com/Budibase/budibase>. (Accessed on 2024-09-16).

- Business Research (2024). Application (large businesses, small businesses, colleges and universities, government, non-profits), regional insights and forecast to 2031. (Accessed on 2024-09-16).
- Caspio (2024). Caspio: Low-Code Platform - Build Online Database Apps. <https://www.caspio.com/>. (Accessed on 2024-09-16).
- CECE (2017). Informatics Education in Europe: Are We All In The Same Boat? Technical report, Association for Computing Machinery, New York, NY, USA.
- db2rest (2024). Github: kdhrubo/db2rest . <https://github.com/kdhrubo/db2rest>. (Accessed on 2024-09-16).
- Eggert, S. and Meier, J. (2010). ERP-Marktüberblick–107 Systeme im Vergleich. *ERP-Management*, 6(3):48–55.
- Elshan, E., Dickhaut, E., and Ebel, P. (2023). An investigation of why low code platforms provide answers and new challenges. In *Hawaii International Conference on System Sciences (HICSS)*, Maui, Hawaii.
- Florescu, D., Levy, A., and Mendelzon, A. (1998). Database techniques for the world-wide web: A survey. *ACM Sigmod Record*, 27(3):59–74.
- Grooss, O. F. (2024). Digitalization of maintenance activities in small and medium-sized enterprises: A conceptual framework. *Computers in Industry*, 154:104039.
- Haug, A., Graungaard Pedersen, S., and Stentoft Arlbjörn, J. (2011). It readiness in small and medium-sized enterprises. *Industrial Management & Data Systems*, 111(4):490–508.
- Herbsleb, J. D. and Grinter, R. E. (1999). Splitting the organization and integrating the code: Conway’s law revisited. In *Proceedings of the 21st international conference on Software engineering*, pages 85–95.
- Hossain, M. (2014). *CORS in Action: Creating and consuming cross-origin APIs*. Manning.
- Hyrnsalmi, S. M., Rantanen, M. M., and Hyrnsalmi, S. (2021). The war for talent in software business-how are finnish software companies perceiving and coping with the labor shortage? In *2021 IEEE International Conference on Engineering, Technology and Innovation (ICE/ITMC)*, pages 1–10. IEEE.
- Kerwin, M. (2017). The ”file” URI Scheme. RFC 8089.
- May, W. (1999). Information extraction and integration with flord: The mondial case study. Technical report, Citeseer.
- Minoli, D. (2008). *Enterprise architecture A to Z: frameworks, business process modeling, SOA, and infrastructure technology*. Auerbach Publications.
- Oracle (2020). Java Client Roadmap Update. <https://www.oracle.com/technetwork/java/javase/javaclientroadmapupdateev2020may-6548840.pdf>. (Accessed on 2024-09-16).
- Oracle (2024). Local Naming Parameters in the tnsnames.ora File. <https://docs.oracle.com/en/database/oracle/oracle-database/23/netrf/local-naming-parameters-in-tns-ora-file.html>. (Accessed on 2024-09-16).
- Peyrott, S. (2024). JWT Handbook. <https://auth0.com/resources/ebooks/jwt-handbook>. (Accessed on 2024-09-16).
- Prinz, N., Rentrop, C., and Huber, M. (2021). Low-code development platforms-a literature review. In *AMCIS*.
- Reese, G. and McLaughlin, B. (2003). *Java Database Best Practices*. O’Reilly & Associates, Inc., USA.
- Richardson, L., Ruby, S., and Demmig, T. (2007). *Web-Services mit REST*. O’Reilly.
- Statista (2024). IT-Fachkräfte nach Qualifikation 2023 — Statista. <https://de.statista.com/statistik/daten/studie/166070/umfrage/veraenderung-der-qualifikationen-von-it-spezialisten-in-itk-unternehmen/>. (Accessed on 07/16/2024).
- StB (2024). Small and medium-sized enterprises (SME) - German Federal Statistical Office. <https://www.destatis.de/EN/Themes/Economic-Sectors-Enterprises/Enterprises/Small-Sized-Enterprises-Medium-Sized-Enterprises/ExplanatorySME.html>. (Accessed on 07/11/2024).
- Uittenbogaard, B., Broens, L., and Groen, A. J. (2005). Towards a guideline for design of a corporate entrepreneurship function for business development in medium-sized technology-based companies. *Creativity and innovation management*, 14(3):258–271.
- van der Schee, M. (2024). PHP-CRUD-API. <https://github.com/mevdschee/php-crud-api>. (Accessed on 2024-09-16).
- Web (2024). HTML Living Standard. <https://html.spec.whatwg.org/dev/>. (Accessed on 2024-09-16).
- Webflow (2024). Webflow: Create a custom website — Visual website builder. <https://webflow.com/>. (Accessed on 2024-09-23).