GETTING STARTED WITH LOW-CODE – A DATA-CENTRIC PRIMER FOR ORACLE APEX[1]

WIELAND SCHWINGER, WERNER RETSCHITZEGGER,
ELISABETH KAPSAMMER, BIRGIT PRÖLL
Johannes Kepler University Linz (JKU), Linz, Austria
wieland.schwinger@jku.at, werner.retschitzegger@jku.at, elisabeth.kapsammer@jku.at,
birgit.proell@jku.at

Abstract The demand for personnel being able to develop Web apps has grown tremendously. Not least to cope with this need, a plethora of *"Low-Code Platforms" (LCPs)* emerged, empowering *"citizen developers"* to build up Web apps without programming skills while enhancing productivity by removing repetitive and boring programming tasks. The *comprehensive functionality* of full-fledged LCPs allowing to specify every nitty gritty detail of a Web app, however, *hampers their adoption*. This has sparked research projects like our EU Erasmus+-project BeeAPEX, cutting a path through the feature jungle of LCPs and lowering the entry barrier for citizen developers. Based on these findings and by focusing on Oracle APEX as representative example, this paper puts forward (i) a *systematic overview of low-code features* to develop the main building blocks of Web apps, (ii) shade light on *nature* and *determining characteristics* of their *development process* and (iii) emphasize on *reuse potential* exploited by LCPs.

Keywords:
low-code platforms, citizen developers, web apps, database systems, Oracle APEX, reuse-driven development

[1] The authors would like to acknowledge the support given by the European Commission through the Action Erasmus + *Better Employability for Everyone with APEX (project ID 2021-1-SI01-KA220-HED-000032218)*, co-funded by the Erasmus+ program of the European Union. The European Commission support for the production of this publication does not constitute endorsement of the contents which reflects the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

University of Maribor Press
DOI https://doi.org/10.18690/um.fov.3.2023.74
ISBN 978-961-286-722-5

1 Introduction

Demand for Web App Developers Increases. The IT sector is one of the fastest growing ones, being key to a green, digital and resilient economy. According to McKinsey, digital adoption in Europe has jumped from 81% to 95% because of the Covid-19 crisis (Fernandez, 2020). Thus, the *demand* for staff with IT skills, e.g., for *developing Web apps,* has also *grown tremendously*.

Low-Code Platforms for Citizen Developers Emerged. Reflecting on this unmatched demand and the growing complexity of Web apps, a plethora of *low-code platforms (LCPs)* have emerged partly stemming from prominent players like Microsoft, Google and Oracle (Bock et al., 2021). Thereby, LCPs substantially draw on *model-driven development principles*, a topic of software engineering researchers for decades (Di Ruscio et al. 2022, Kapsammer et al. 2017, Retschitzegger et al., 2015, Schwinger et al. 2021). *"Low-code"* means that only few or even no programming skills are required. Thus, LCPs *empower* non-IT-experts, i.e., *"citizen developers",* making Web app development attractive to a large number of people (Luo et al., 2021) and enhance *productivity* by removing repetitive programming tasks (Bock et al., 2021). Gartner forecasts that LCPs will account for 65% of all Web app developments by non-experts in 2024 (Mehta, 2022).

Feature Overload Hampers Adoption by Citizen Developers. Despite these benefits, the *comprehensive functionality* of full-fledged LCPs like Oracle APEX providing sophisticated tooling to specify every nitty gritty detail of a Web app might actually *hamper their adoption* by citizen developers (Mussbacher et al., 2021).

There are already efforts to *pin down the essence of* LCPs (Bock et al., 2021, Farshidi et al., 2021, Lichthentäler et al., 2022, Sahay et al., 2020) mostly *targeting*, however, *IT experts only*. At the same time, research projects have been sparked like *our EU Erasmus+-project BeeAPEX*, cutting a path through the feature jungle of LCPs for *non-IT-experts*, thus lowering the barrier for citizen developers.

Paper Contribution and Structure. Based on the findings in BeeAPEX, the overall contribution herein is a primer for getting started with LCPs from a non-IT-expert perspective. Focusing on Oracle APEX as representative example, observations are generalizable across *DB-centric LCPs* (Bock et al., 2021) as we (i) put forward a *systematic overview of low-code features* to develop the main building blocks of Web apps, (ii) shade light on the *nature* and the *determining characteristics* of their overall *development process* and (iii) emphasize on *reuse potential* exploited by such LCPs based on existing data and Web apps. For this, Section 2 discusses *Oracle APEX from a bird eyes view*, Sections 3, 4 and 5 deal with *DB layer*, *Web layer* and *data exchange* and finally Section 6 focuses on *future work*.

2 Oracle APEX from a Bird Eyes View

Broad Application Domains and Business Needs – Reuse Crucial. *Oracle APEX (Application Express)* is a LCP for Web apps based on an Oracle Database Management System (DBMS) (Sciore, 2020). It is employed by large and small customers alike, across a *broad number of application domains*, coping with a *wide spectrum of business needs* (Baggia, 2018). These may range from simple transformations of local spreadsheets into Web-based ones, to full-fledged Web apps storing, retrieving, processing and visualizing business transactions (Baggia, 2019). Independent of complexity, domain or business needs targeted by LCPs in general and Oracle APEX in particular, *reuse of already existing artifacts* is crucial.

Reuse-Driven DB Layer and Web Layer. For this, APEX offers Web browser-based tools along with "wizards" assisting to build complete Web apps comprising DB layer and Web layer (Retschitzegger et al., 2009), (cf. Figure 1).

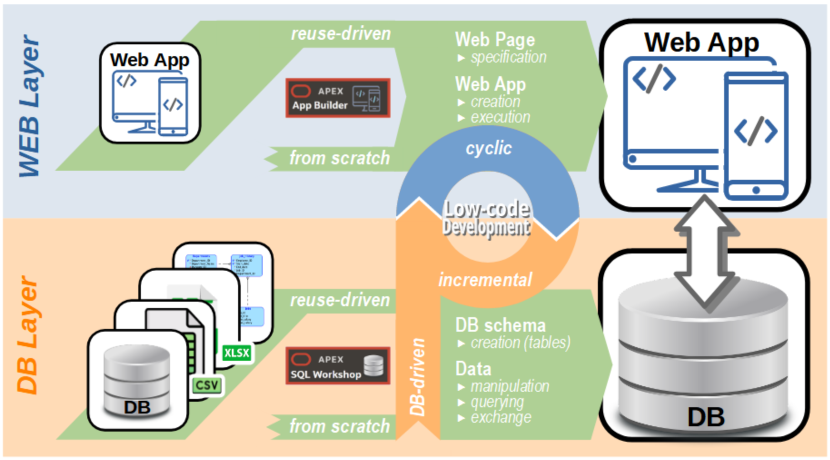

Figure 1: Reuse-Driven Development of DB Layer and Web Layer

For the *DB layer*, first, the *DB schema* to be specified, in case of *relational DBs* in terms of *tables* and second, the actual *data* needed for the Web app has to be provided (cf. Section 2). Both can be done either *manually from scratch* or ideally, by *reusing existing schemata and/or data* in terms of *offline import/export* or *online data exchange* (cf. Section 4). Generally speaking, tools like *APEX "SQL Workshop"* address these tasks. Regarding the *Web layer* which builds upon the DB layer, citizen developers are faced with the *specification of Web pages* and the automatic *creation* and *execution* of the final *Web app* (cf. Section 3). This process can be again either start *from scratch* or by reusing an *existing Web app or parts thereof*. These tasks are supported by *APEX's "App Builder"*. Finally, it has to be noted that not only data and their schema are stored within the DB layer, but also all the artifacts making up the Web app (e.g., pages, logics and data access).

Cyclic and Incremental Low-Code Development Process. While APEX provides a set of tools coming with a series of wizards to assist the developer in specifying both, DB layer and Web layer, it does not imply, however, a single linear development process, being rather *cyclic* and *incremental*. This is in line with predominance of *agile processes* in Web app development, making it, however, getting started for citizen developers even harder (Bucchiarone et al., 2021). Thus, in the following, we elaborate the overall Web app *development options* focusing on processes from a *navigation perspective* through the APEX tooling as well as from a *reuse-driven and data-centric perspective*.

3 Getting Started with the DB Layer

DB Layer Development Steps. Getting started with the DB layer, roughly speaking, at the very end, it's all about creating appropriate *DB tables* in terms of a *DB schema* (cf. Step 1 in Figure 2), *storing* and *manipulating data* (cf. Step 2) and *querying data* (cf. Step 3). In the following, the main focus will be on Step 1, not least since being the most complex task which is therefore appropriately supported by APEX through different low-code development options.

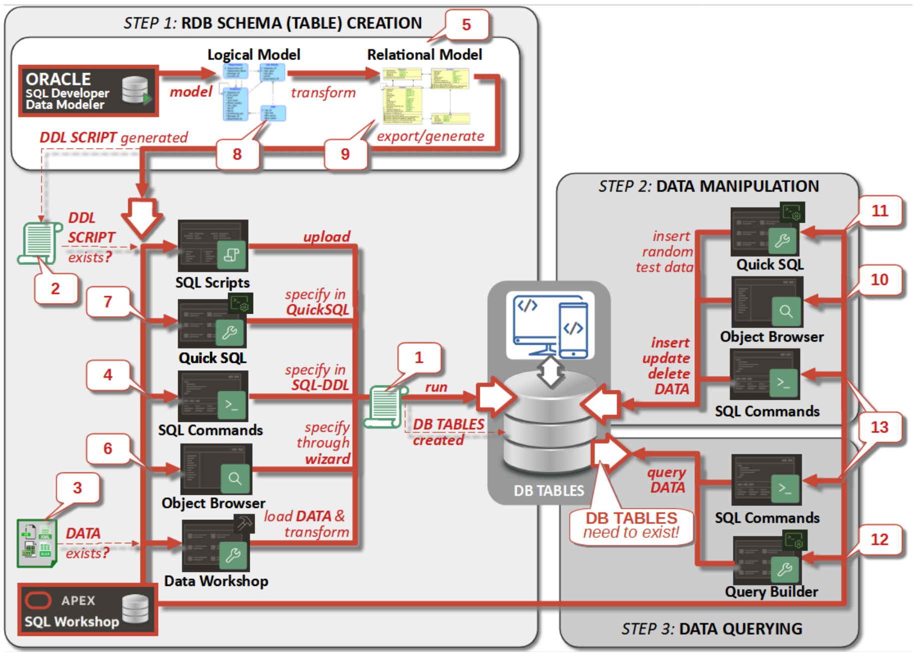

Figure 2: Overall Process for Creating the DB Layer.

3.1 DB Schema Creation – Step 1

DB Schema Creation From Scratch or Reuse-Driven. For DB schema creation, *different options* exist as depicted on the left-hand side of Figure 2. Overall, these options can be distinguished whether they support schema creation *from scratch* along different *low-code abstraction levels* or if they allow for *reuse*. No matter, which option is chosen, ultimately, a so-called *DDL (Data Definition Language)-script* containing programmatic "CREATE TABLE"-commands is needed for the actual creation of *empty DB tables* (cf. #1 in Figure 2).

Reuse-Driven Schema Creation. When developing a DB schema, maximizing reuse should have, of course, priority whereby in the simplest case the *DDL-script could be already available* (cf. #2 in Figure 2). This script can be simply imported via *Oracle SQL Workshop* and executed, leading to the ultimate goal of empty DB tables. The second reuse option is that *existing data within external files (e.g., XLSX files)* can be used as basis for automatic DB table generation (cf. #3 in Figure 2). Thereby, *schema information* (e.g., attributes) is *automatically inferred* and can be augmented by user information provided through a wizard, which is, however, limited to quite simple tables only (cf. Section 4).

Low-Code-Driven Schema Creation. In case that reuse is not possible, there are again different options for developing the DB tables *from scratch*, either using the *traditional programmatic way* via *SQL commands* (cf. #4 in Figure 2) or employing a more *low-code development style*, where the DDL-scripts are automatically generated out of other more *abstract artifacts*, manually specified by citizen developers. Here we can distinguish three different options, depending on the abstraction level where the development takes place, ranging from *graphical model-driven schema creation* (cf. #5 in Figure 2), to *wizard-driven schema creation* (cf. #6 in Figure 2) and finally, at a more concrete level, to a *short-hand SQL-like schema creation* (cf. #7 in Figure 2). Depending on the abstraction level to start off, different concepts need to be understood and different tools are employable. In the following, these options are discussed, ordered along their abstraction level.

(1) Model-driven Schema Creation. Ideally, development shall start in a low-code fashion by *graphically modelling the DB schema* thus being "more distant" from code. For this, tools like the *"Oracle Developer Modeler (ODM)"* can be employed, being however, not part of Oracle APEX.

Thereby, in a first development step, a so-called *logical (data) model* can be defined (cf. #8 in Figure 2) to describe "things" of the real world (i.e., the problem domain) in practice often in terms of so-called *Entity-Relationship (ER)-diagrams* (Sciore 2020), focusing on *entities* (i.e., the "things") with their *attributes* (i.e., properties of "things") and *relationships* (i.e., how "things" relate to each other) without considering specifics of a certain DBS. This facilitates understanding and communication within a development team as well as with customers, being of particular benefit for non-IT-experts.

In a second step, this DB-independent ER-model, can be automatically transformed into a DB-specific schema which is also graphically visualized by ODM, called *Relational DB (RDB)-schema,* using the *Relational Model (RM)* as formalism (cf. #9 in Figure 2) to describe the structure of data as a collection of *relations* aka. *tables*, thus resembling entities of the ER-diagram. This RM has to be *manually detailed* by defining *datatypes* for attributes and by expressing *relationships* through so-called *foreign-keys*, being in fact designated attributes acting as pointers to the *identifying (primary key) attribute* of other tables. Finally, out of the RM, ODM allows to automatically generate according DDL-scripts which can be imported into APEX using Oracle SQL Workshop and executed to *automatically generate the DB tables.*

(2) Wizard-driven Schema Creation. Besides the model-driven option, Oracle provides a simple *form-based option* (cf. #6 in Figure 2) to create new tables and modify them if necessary, in a wizard-driven manner. This is ideal for those users having no modelling knowledge, but limits one to the options provided which need to be understood, to capitalize on the full functionality.

(3) Textual, Shortcut-Driven Schema Creation. Being most concrete and already close to the programmatic option is to use a simple textual "shortcut-notation" for SQL, provided by Oracle's *"Quick SQL"* (cf. #7 in Figure 2). This is a good choice if new tables are needed, e.g., for *quick testing* purposes, not least since also random data can be inserted. Quick SQL is, however, a non-standard notation, offering limited expressiveness. Therefore, for more detailed specifications, altering the generated DDL-scripts is necessary.

(4) Programmatic Schema Creation. Finally, there is of course also the possibility of using SQL-DDL in terms of `"CREATE TABLE"`-commands (cf. #4 in Figure 2), providing the benefits that every single table specification detail can be defined as needed and being not dependent on, sometimes sub-optimal automatic generation processes. However, it naturally entails the burden to deal with code and to be familiar with the necessary syntax.

3.2 Data Manipulation and Querying – Steps 2 and 3

Based on the created DB schema, data can be *manipulated*, i.e., inserted, updated or deleted as well as *queried*. Depending on the business needs, data will be managed directly by the Web app (cf. Section 3) or prior to the deployment of the Web app during development of the DB layer. Regarding the latter, for manipulating data, there are again different options, ranging from *low-code wizard-driven* through *Oracle's Object Browser*, allowing data inserts updates and deletes (cf. #10 in Figure 2), over an *auto-generation of random (test)data* through *Quick-SQL* as already mentioned (cf. #11 in

Figure 2), to the *exchange of existing data* (cf. #3 in Figure 2 and Section 4), the latter two, however, allowing for data insertions only.

Regarding the task of *data querying*, there is again a *low-code, wizard-driven* option via *Oracle's Query Builder* available (cf. #12 in Figure 2). For both data management tasks, i.e., manipulation and querying, there exists, of course, also the programmatic option in terms of SQL commands (cf. #13 in Figure 2).

4 Getting Started with the Web Layer

Web Layer Development Steps. Getting started with the Web layer, it is similar to the DB layer all about *reusing existing artifacts*, comprising, specific to the Web layer, (parts of) existing *Web apps*, *DB tables* and *data* (cf. Step 1 in Figure 3) *specifying (additional) Web pages*, mostly *on top of existing DB tables* together with the *navigation* in-between (cf. Step 2) and finally, *automatically creating and running the Web app* by some simple button clicks (cf. Steps 3 and 4). These steps are organized in a *multi-step, partly cyclic manner* allowing for *incremental development*, i.e., a stepwise refinement of the Web app and eventually the DB layer, supported in a low-code fashion by *APEX's "App Builder"*. In the following, an overview of the low-code support provided for the aforementioned steps is given.

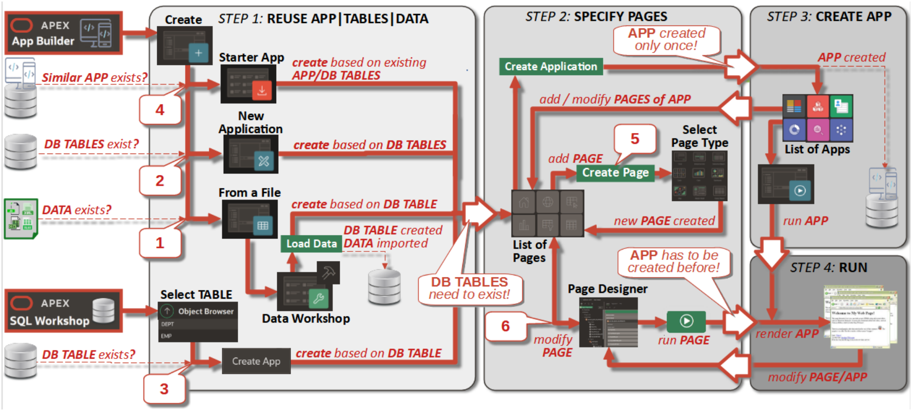

Figure 3: Overall Process for Creating the Web Layer

4.1 Reuse App | Tables | Data – Step 1

Web Layer – From Scratch or Reuse-Driven. First of all, no matter if reuse is possible or not, as APEX focuses on DB-driven Web apps, it is advisable to always start by developing the DB layer first (cf. Section 3). Reuse potential is taken into account by APEX supporting several use cases during low-code development of the Web layer, i.e., *(i) reuse of data:* as already mentioned in Section 3, if data exists in external files, the DB layer can be automatically generated (cf. #1 in Figure 3), *(ii) reuse of DB tables:* if the DB layer (or parts of it) already exists the Web layer can be built directly thereupon (cf. #2 and #3 in Figure 3), *(iii) reuse of Web apps*: if reuse and further modification of an already existing Web app together with its DB-layer is reasonable (cf. #4 in Figure 3).

4.2 Specify Pages – Step 2

Wizard-Driven Page Specification – Selecting Page Types. After the DB layer exists, the main task of Web layer development is to incrementally *add new Web pages* forming its basic building blocks, using the APEX *"Create Page Wizard"* (cf. #5 in Figure 3). Each page can be based on one or more DB table(s), allowing to visualize and manipulate their data. Although these pages are naturally internally specified in terms of code (e.g., HTML), LCPs like APEX initially hide this programmatic layer in that citizen developers can simply choose out of different *page types*, coming with predefined functionality. The provision of such predefined functionality recurrently found in Web apps like *lists*, *reports* and *charts* for *data visualization* as well as *reports* and *forms* for *data manipulation* allows LCPs to capitalize thereupon by generating the necessary code for those Web pages along with code for DB data retrieval and manipulation. Finally, the Web pages *can be linked together* using navigation menus, tabs, buttons, or hypertext links.

Wizard-Driven Page Modification – "Page Designer". Once a page is created, it can be, at any time during low-code development, further maintained and enhanced using *APEX's "Page Designer"* (cf. #6 in Figure 3). By a combination of forms, wizards, and extension points to specify code (e.g., PL/SQL or JavaScript), the composition/layout of pages can be fully modified.

4.3 Create and Run Web App – Steps 3 and 4

Create the Web App. As soon as one or more pages have been created, it is ultimately necessary to initially create the necessary artefacts for the Web app. Thereby, some *overall properties* of the *whole Web app* (e.g., appearance of the app) can be selected which automatically leads to generation of additional functionality of the Web app in a low-code fashion. *Incremental development* is again supported since after creation, additional pages can be added or existing ones modified.

Run the Web App. Finally, the resulting app and/or each of the specified pages can be *interactively tested*, as the resulting HTML-pages are rendered allowing to further refine the Web app. Again, *incremental development* is possible since it can be navigated back to Page Designer after running and testing the Web app.

4 Getting Started with Data Exchange

Data Exchange Development Steps. Getting started with data exchange, it's all about the *import and export of data* which can be done *offline* (cf. Step 1 and Step 2 in Figure 4) and complementary to that, *online access to data for external clients* like other Web/mobile/legacy apps or cloud-based services (cf. Step 3). It has to be noted that overall, data exchange is a *cross-cutting concern*, again emphasizing on the *reuse aspect* when developing Web apps on basis of LCPs. In the following, an overview of the APEX low-code support for these three steps will be given.

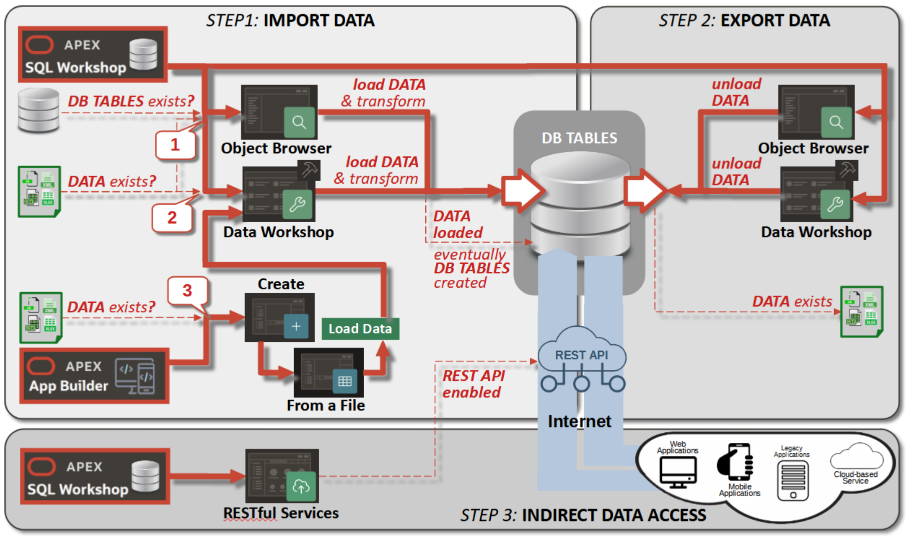

Figure 4: Overall Process for Data Exchange

5.1 Offline Data Import and Export – Steps 1 and 2

Two Import | Export Alternatives – "Object Browser" | "Data Workshop". The simplest low-code option for data reuse in terms of data import/export from/to files is using *SQL Workshop's "Object Browser"*, being limited, however, to a *table-wise processing* (cf. #1 in Figure 4). In contrast, *Oracle's "Data Workshop"*, allows imports/exports based on *several tables*, being especially suited for data of a moderate size (fewer than 10 tables) having standard datatypes only, i.e., no multi-valued fields or nested structures. For importing/exporting *huge and complex data*, other tooling, such as "*SQL*Loader Utility*" is more suitable, requiring, however programming skills. Data Workshop *can be accessed* for import and export via SQL Workshop (cf. #2 in Figure 4) and additionally, for import only, via App Builder when intertwinedly creating Web apps (cf. #3 in Figure 4).

Supported File Formats. Regarding possible file formats for data reuse, for both, import and export, any standard *delimited format (e.g., CSV – Comma Separated Values)* as well as *XML files (eXtended Markup Language)* are allowed. Just for imports, two additional formats are supported, comprising *XLSX files (i.e., Excel workbooks)* and *JSON files (Java Script Object Notation)*.

Wizard-Driven Importing Steps. For importing data for the sake of reuse, Data Workshop provides a *"Load Wizard",* guiding citizen developers through all necessary import steps. The two main steps comprise *(1) provision of the data source* via drag and drop or a load-file dialogue and *(2) configuration of the Data-to-Table mapping.* The latter allows to decide if data should be loaded into a *new table* (which can be automatically created based on the structure of the file to be imported) or into an *existing one* and to define the *mapping* between the *columns* of the source file and those of the table. After activating the load process via a *"Load Data"-button*, the data is actually loaded whereby the *loading dialog* informs how many rows have been loaded. The resulting table can now be viewed via Object Browser.

Wizard-Driven Exporting Steps. For exporting data for reuse, *file format*, *table* and *columns* have to be selected whose data should be exported. In case of a *delimited format*, additionally (i) the delimiter between rows has to be defined, (ii) if row names should be included in the output and (iii) if the data format should be DOS or UNIX, before finally *"Unload Data"* to save the export (cf. Figure 4).

5.2 Online Data Access – Step 3

The Theory Behind – "REST Architectural Pattern". Exchanging data by enabling external clients (e.g., Web/mobile/legacy apps) to reuse data by online access, can be easily realized using *RESTful services*, again in a low-code fashion. *REST (Representational State Transfer)* is an *architectural pattern for interoperability* between arbitrary systems over the Internet (Fielding et al, 2010). It enabling data querying/manipulation *without the need for direct access* to the underlying tables. **Realizing Indirect Access via "RESTful Services"**. For realizing data reuse according to the REST pattern, *RESTful services* have to be created on top of DB tables, i.e., *"REST-resources".* These are identified by URLs and accessed over `HTTP` or `HTTPS`, requesting one of four different kinds of operations – `POST`, `GET`, `PUT` and `DELETE` provided by the *REST API (Application Programming Interface)* – thereby resembling the well-known *CRUD-operations* (Create, Read, Update, Delete) and thus detailing in which way data reuse is possible through indirect access. Such requests to a RESTful service always elicit a *response* in the form of XML, JSON, HTML, or some other standard format (cf. Figure 5).

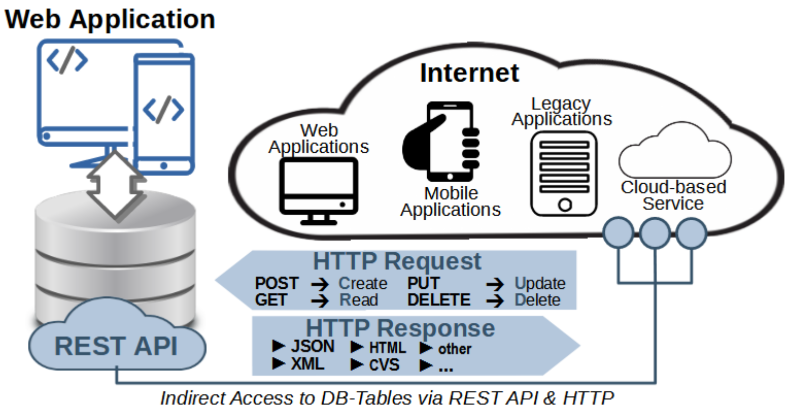

Figure 5: RESTful Services for Online Data Access – Basic Architecture

Low-Code based Definition of RESTful-services – "AutoREST". For each DB operation being offered to external clients as RESTful service for indirectly accessing a DB table, several steps would be necessary, being beyond the scope of this article. There is, however, for the simplest form of a query, i.e., a *full table scan* having a *fixed output format in terms of JSON*, a low-code option provided in terms of APEX's *"AutoREST"-feature*. There are only two simple steps necessary, *(1) Enabling the DB Schema for RESTful access* via SQL Workshop's *RESTful services tool* and *(2) Defining a table as RESTful resource* using Object Browser. After that, REST has been activated for the table and the *automatically generated access URL* appears allowing the service to be simply tested in a Web browser.

6 Future Work

In this paper, a first step has been taken towards pinning down the essence of full-fledged DB-centric LCPs for non-IT-experts. Based on this foundation, we are currently conceptualizing in the course of our Erasmus+ project "BeeAPEX" twelve different business use cases, ranging from a simple Web shop to systems for course scheduling, team appointment, car rental and diet management. Each of these use cases is described in detail comprising business view, problem definition, use case diagram, data model and a step-by-step guide for the realization on basis of Oracle APEX. Thus, by combining our guide how to get started with LCPs with these practical use cases, we intend to further lower the barrier for non-IT-experts to get engaged in low-code development.

Acknowledgements

We owe a big debt of gratitude for the in-depth discussions about Oracle APEX to the partners in our Erasmus+-project "BeeAPEX", comprising University of Zagreb, University of Maribor, University of Žilina Slovakia, International Hellenic University and Kosminsky University Warsaw.

References

Baggia, A., Mali, A., Grlica, A., Leskovar, R. (2018). Oracle APEX in Higher Education. 37th Int. Conf. on Organizational Science Development, Portorož, Slovenia.

Baggia, A., Leskovar, R., Blaž Rodič, (2019). Low-code Programming with Oracle APEX offers new Opportunities in Higher Education. 3rd Int. Scientific Conf. Recent Advances in IT, Tourism, Economics, Management and Agriculture (ITEMA), Bratislava, Slovakia.

Baggia, A., Leskovar, R., Rajkovič, U., Motušić, A. (2022). Low-code programming and web application development. Information Society, Ljubljana, Slovenia.

Bock, A., Frank, U. (2021). In Search of the Essence of Low-Code: An Exploratory Study of 7 LCPs. ACM Int. Conf. on MDE Languages and Systems (MODELS), Fukuoka, Japan.

Bucchiarone, A. et al. (2021). What Is the Future of Modeling? IEEE Software, vol. 38, no. 2.

Di Ruscio, D., Kolovos, D., de Lara, J. et al. (2022). Low-code development and model-driven engineering: Two sides of the same coin?. Software System Modeling (SoSyM 21), Springer.

Farshidi, S., Jansen, S. & Fortuin, S. (2021). Model-driven development platform selection: four industry case studies. Software System Modeling (SoSyM 20), Springer.

Fernandez, S., Jenkins, P., Vieira, B. (2020). Europe's digital migration during COVID-19: Getting past the broad trends and averages, Mc Kinsey Digital.

Fielding, R., Taylor, R. (2010). Principled design of the modern Web architecture. Edited by Mehdi Jazayeri and Alexander L. Wolf Carlo Ghezzi. ACM, ISBN: 978-1-58113-206-9.

Kapsammer, E., et. al. (2017). On the Evolution of Modeling Ecosystems: An Evaluation of Co-Evolution Approaches. Proc. of the 5th Int. Conf on Model-Driven Engineering and Software Development (MODELSWARD), Porto, Portugal.

Lichtenthäler, R., et. al. (2022). A Use Case-based Investigation of LCPs. Proc. of the 14th ZEUS Workshop on Services and their Composition, Bamberg, Germany, CEUR-WS.

Luo, Y., et al. (2021). Characteristics and Challenges of LCD: The Practitioners' Perspective. In Proc. of the 15th ACM Int. Symp. on Empirical SWE and Measurement (ESEM).

Mehta, V. (2022). Forecast Analysis: Low-Code Development Technologies Worldwide. Gartner.

Mussbacher, G., et al. (2021). A Hitchhiker's Guide to Model-Driven Engineering for Data-Centric Systems, in IEEE Software (38, 4).

Retschitzegger, W., et al. (2015). Model-Driven Co-evolution for Agile Development. 48th Hawaii Int. Conf. on System Sciences (HICSS), Kauai, Hawaii, USA.

Retschitzegger W., et al. (2009). Web Engineering: The Discipline of Systematic Development of Web Applications, Wiley. ISBN: 978-8-1265-2162-3

Sahay, A., et al. (2020). Supporting the understanding and comparison of LCPs. 46th Euromicro Conf. on SWE & Advanced Applications (SEAA), Portoroz, Slovenia.

Schwinger, W, et al. (2021). Behavioral Interfaces for Executable DSLs. In: Koziolek, A., Schaefer, I. & Seidl, C. (Hrsg.), Software Engineering, Bonn: GI e.V.

Sciore, E. (2020). Understanding Oracle APEX 20 application development. 3nd ed. Apress. ISBN: 978-1-4842-6165-1