BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

APPLICATION FOR APPROVING BANK PAYMENTS ON S/4 HANA SYSTEM

APLIKACE PRO SCHVALOVÁNÍ BANKOVNÍCH PLATEB V SYSTÉMU S/4 HANA

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR Bc. PETR BEČKA
AUTOR PRÁCE

SUPERVISOR Doc. Ing. JAROSLAV ZENDULKA, CSc.
VEDOUCÍ PRÁCE

BRNO 2018

Master's Thesis Specification/21075/2017/xbecka02

Brno University of Technology - Faculty of Information Technology

Department of Information Systems Academic year 2017/2018

Master's Thesis Specification

For:	**Bečka Petr, Bc.**
Branch of study:	Information Systems
Title:	**Application for Bank Payment Approval in SAP S/4HANA**
Category:	Information Systems

Instructions for project work:

1. Get acquinted with workflow and business process management and its application for bank business approval and related payment processes.
2. Analyze requirements for application that will support these processes.
3. Explore available frameworks for the development of web application in JavaScript and based on agreement with you supervisor choose one for implementation.
4. Design the application.
5. Implement the application using the choosen framework.
6. Verify the functionality on a suitable sample of data.
7. Summarize achieved results and discuss possible further development.

Basic references:

- Workflow Management Coalition. http://www.wfmc.org/.
- Okungbowa, A.: SAP ERP Financial Accounting and Controlling. Apress, 2015, 596 p.
- Bavaraju, A.: SAP Fiori Implementation and Development. SAP Press, 2nd Edition. 2017, 614 p.
- Bönnen, C. at al.: OData and SAP Gateway. SAP Press, 2nd Edition, 2016, 780 p.

Requirements for the semestral defense:
Items 1 to 4.

Detailed formal specifications can be found at http://www.fit.vutbr.cz/info/szz/

The Master's Thesis must define its purpose, describe a current state of the art, introduce the theoretical and technical background relevant to the problems solved, and specify what parts have been used from earlier projects or have been taken over from other sources.

Each student will hand-in printed as well as electronic versions of the technical report, an electronic version of the complete program documentation, program source files, and a functional hardware prototype sample if desired. The information in electronic form will be stored on a standard non-rewritable medium (CD-R, DVD-R, etc.) in formats common at the FIT. In order to allow regular handling, the medium will be securely attached to the printed report.

Supervisor:	**Zendulka Jaroslav, doc. Ing., CSc.**, DIFS FIT BUT
Consultant:	Čus Ľuboš, Ing., SAP
Beginning of work:	November 1, 2017
Date of delivery:	May 23, 2018

L.S.

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

[illegible]

Dušan Kolář
Associate Professor and Head of Department

Abstract

The main goal of this thesis is to develop user interface and database queries to gain necessary business data for an application for the approval of bank payments using modern technologies and technologies provided by SAP company. The first part describes bank payment issues in general, key concepts are explained and some problems are mentioned from this area, with which companies struggle today. The SAP S/4HANA architecture, for which the application is developed, is also described. Last but not least, there are mentioned technologies for creating user interfaces that are suitable for this development or are one of the most used today. The application design process describes its requirements, the creation of a prototype application that visualizes the way the user sees the data and the design of the data model. The resulting application allows you to retrieve the correct data from the database, display it in a predefined way, and offers the option of editing and managing it.

Abstrakt

Hlavním cílem této práce je vyvinout uživatelské rozhraní a získávání nezbytných podnikových dat v aplikaci sloužící pro schvalování bankovních plateb za použití moderních technologií a technologií poskytovaných společností SAP. V první části je obecně popsáno téma bankovních plateb, jsou vysvětleny klíčové pojmy a zmíněny některé problémy z této oblasti, se kterými se společnosti a soukromníci v dnešní době potýkají. Dále je popsána architektura SAP S/4HANA, pro kterou je aplikace vyvíjena. V neposlední části jsou zmíněny technologie pro tvorbu uživatelských rozhraní, jenž jsou pro tento vývoj vhodné anebo jsou dnes nejvíce používány. Proces návrhu aplikace popisuje její požadavky, tvorbu prototypu aplikace, jenž vizualizuje způsob zobrazení dat uživateli a návrh datového modelu. Výsledná aplikace umožňuje získávat správná data z databáze, zobrazovat je definovaným způsobem způsobem a nabízí možnost jejich editace a správy.

Keywords

JavaScript, SAPUI5, React, Angular 2, Vue.js, S/4 HANA, SAP HANA, OData, SAP Gateway, web, testing

Klíčová slova

JavaScript, SAPUI5, React, Angular 2, Vue.js, S/4 HANA, SAP HANA, OData, SAP Gateway, web, testování

Reference

BEČKA, Petr. *Application for approving bank payments on S/4 Hana system.* Brno, 2018. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Doc. Ing. Jaroslav Zendulka, CSc.

Application for approving bank payments on S/4 Hana system

Declaration

Hereby I declare that this diploma's thesis was prepared as an original author's work under the supervision of Mr. Doc. Ing. Jaroslav Zendulka, CSc. The supplementary information was provided by Mr. Ing. Luboš Čus All the relevant information sources, which were used during preparation of this thesis, are properly cited and included in the list of references.

. . . .
Petr Bečka
May 23, 2018

Acknowledgements

I would like to thank my superior Mr. Doc. Ing. Jaroslav Zendulka, CSc., who was very helpful, understanding and willing during the development of the thesis. He was very supportive and calm during the whole process.

I would also like to thank SAP employees for the time they gave me during the discussions on effective problem solving and help with searching for information, namely my consultant Mr. Ing. Luboš Čus and my colleagues Mr. Ing. Norbert Volf and Mr. Ing. Michal Uhlíř.

I would also like to thank my always supporting family and girlfriend, because of them I have always had the time and the determination to do everything.

Contents

Chapter 1

Introduction

Approval of bank payments is a very important process that is now practiced in large, medium but even in a smaller companies. In the end, without such a measure, a company employee could generate payments for their or other own accounts. Payments would then be unchecked and fraudulent within the company. This process can be controlled by the authorized person using predefined documents, which, according to some pre-arranged steps, could confirm payments. In large companies that generate large amounts of payments, the whole department can be set up for this purpose. Even in this case, the process of approving bank payments is opaque and risky. Application for the approval of bank payments will serve to fully automate this process of controlling the payments made, and to increase the transparency. It will also provide an overview of the company's over all payments and reduce the possibility that they will be lost or omitted.

In this diploma thesis will be in following chapters described the issue of generating payments on accounts payable. There will be explained what the accounts payable is and what it represents for the company and why it is important to have and keep a good overview of them. Plus there will be mentioned how this process works in SAP environment.

In chapter about SAP S/4HANA will discuss and represent the components of this architecture on which the application will be deployed. There will be mentioned their importance and the role they play within architecture, the way in which each component is connected and how they communicate between each other.

The chapter on front-end frameworks will present the strengths and weaknesses of those of today's most powerful and most widely used frameworks. The aim of this chapter is to evaluate which technology will the most suitable to implement such an application.

Then, the chapter titled application for approving bank payments will discuss the process of developing the entire application, including collection of requests and design of the user interface. Besides, the basic application architecture will be discussed here, and then the necessary adaptation of the architecture of this particular application will be explained. It will be mentioned in more detail how application basic and necessary functionality were implemented. It will also describe how the use of SAPUI5 framework libraries, that provide time consuming functionality for implementation, have been tailored to the application.

The last chapter will show how the testing was done. Examined will be the types of tests that have been performed, their purpose, and the results that developers provide.

Chapter 2

Accounts Payable

This chapter provides a general introduction into the topic about obligations from the accounting point of view, how it looks like when someone is processing payment for approval from the time when commitment has been created up to the time where payment is successfully approved and debit is sent to the bank institution. Last but not least, there will be introduced solutions that are frequently used and available for SAP's customers.

2.1 Accounts Payable area

Accounts payable are huge and important part of the company's liabilities. It is therefore necessary to keep and record them in the company's accounting tables for an evidence. In the resulting register, accounts payable are shown in the balance sheet and divided into short-term and long-term categories.

Accounts payable are created on the basis of some events that happened in the past and are expected to result into any expense on the part of the company to cover such a fact. These resources can be both monetary, tangible and / or intangible. These obligations are usually legally enforceable by signing a binding contract.

In the accounting area, it is a legal obligation to have all the liabilities saved and correctly recorded, they are then shown in the balance sheet. A short-term commitment is expected to be paid within less than 12 months. Otherwise, we are talking about a long-term commitment.

In the end of the day, if company doesn't want to pay for things or services that were not ordered and received, it is necessary to keep an eye on and keep track of the payables and receivables.[18]

2.2 Accounts Payable function

Business accounting departments often have more things to do in their job description than just paying bills and incoming invoices. Especially in larger companies are accounts payable employee mostly responsible for accounting payments, while in a smaller businesses are account payable and accounts receivable very often connected. Accounts payable could be divided into some basic categories, namely it is vendor payments, internal payments, travel expenses and other payments.[16]

2.2.1 Vendor Payments

Processing incoming invoices and paying suppliers' obligations is the first thing that comes to mind under the accounting department term. Vendor payments include monthly end-of-month debt analysis and reports showing how the business is running and how much the business currently owes to suppliers.

2.2.2 Internal Payments

Accounts payables also work internally, they usually divide intra-company payments. Internal reimbursement costs are managed by predefined business management procedures. These procedures determinate that employees should process a manual report, confirmations or substantiate claims for payments. Smaller amounts are mostly used for smaller cash expenses such as postage, office supplies, or lunches. Accounting payables are then used to gain sales on taxes for a department managers when purchasing the necessary equipment to ensure the smooth running of their business.

2.2.3 Business Travel Expenses

This fact is very common in larger enterprises where owners and / or employees travel more frequently. Accounts payable department then can add travel costs and local payments on the go between accounts. Internal audits may require for example travel expenses on which they can provide funding to employees. After they come back, the due accounts are compared with the provided funds compared to the actual expenses.

2.2.4 Other Payments

The acounts payable department also works hard to reduce costs by paying attention to details that can save business money, for example, when an invoice is paid within the discount period, that is provided by suppliers, it is many times the front contact between sales representatives and dealer representatives.

As a result, building and maintaining good relationships often falls into accounts payable. Strong and healthy relationships can beneficial for the business when working with suppliers. For example during a seasonal sales decline a good relationship may lead to loose credit conditions.[20]

2.3 Payment Process in general

The method or function of accounts payable is extremely important because it includes almost all company payments except the employees salaries. The whole accounting process can be carried out by the accounting department within a large company, a small personnel department in a medium-sized company, by an accountant or a business owner in small-sized company.[14]

Regardless the size of the company, the responsibility of the accounting department is to pay the statutory and precise defined company's invoices. This means that the invoice itself must reflect information about what the company ordered and received, correct unit costs, calculations, sums, conditions, etc.

To make sure that the funds and other assets of the company are processed correctly, the process of accounts payable should be a subject of internal control. The reason for introducing internal control may be to prevent from the payment of a fraudulent invoice, the payment of an inaccurate invoice, or to prevent multiple payment of the supplier's invoice.

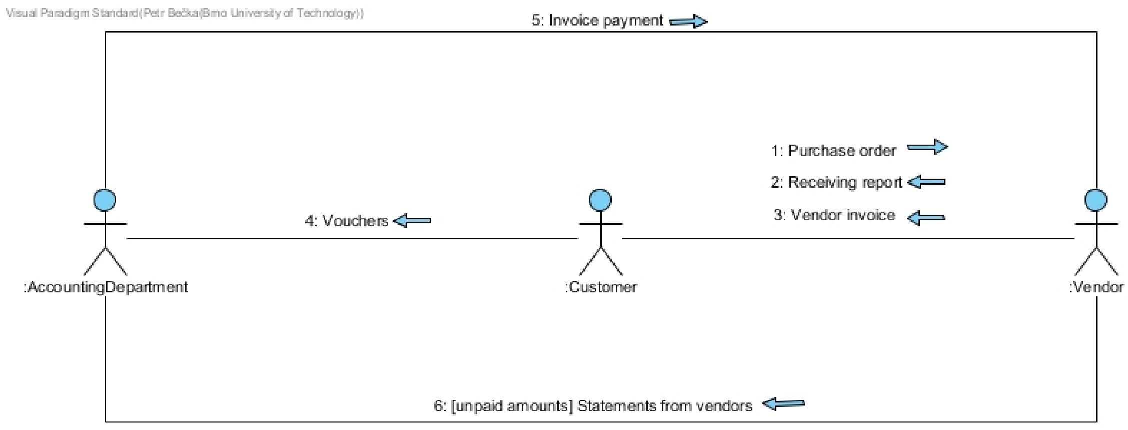


Figure 2.1: Communication diagram showing the sequence of individual steps from creating an order up to invoice payment.

The due date of the accounting obligations must be also effective and accurate so that the company's accounts could be accurate and complete as well. Due to double-entry bookkeeping, forgetting of the supplier's invoice may eventually lead into two accounts to report incorrect amounts. For example, if the repair costs are not recorded on time, the liability will be removed from the balance sheet and repair costs will be omitted from the profit and loss statements. If the supplier's invoice for repair is recorded twice, there are also issues coming up. First, commitments will be overestimated, and repair costs will be overestimated as well. To tell it different, without having billing process well observed and organized, management and the other users of the financial statements will receive inaccurate feedback on the performance and financial position of the company.

Poor evidence of billing could also mean a missing discount for too early payed debt. If suppliers' invoices are not payed at the due date time, the relationships with vendors could go cold. This may lead to the situation sellers requiring charge for a cash on delivery . If such a situation occurs, it could have extreme consequences for the company's existence.

As well as delays in paying bills, early payment of accounts payable may also lead into problems. If the supplier's invoices are paid earlier than necessary, then sufficient budget may not longer be available, which may cause that other invoices could not be paid within the due date. Each bank payment is also supported by documents like purchase order, receiving report, vendor invoice, vouchers and statements from vendors that all confirm the validity and authorization of such payment. Process of receiving such a documents could be seen on figure 2.1.

In general, the documents that are generated during this process are preserved and pinned to payments to verify the validity its for later reimbursement. Usually they are a purchase order, a receipt report and a vendor invoice.

2.3.1 Purchase order

The resulting purchase order is a document prepared by the company to accurately disclose and record what the company orders from the vendor. The paper version of the order is a form with copies distributed to the several people. People or departments that receive a copy of the purchase order include:

- person requesting the issuance of a purchase order for goods or services,
- department that is required to pay the order,
- receiving department,
- suppliers,
- person who prepares the order.

All the information as the purchase order number, date of issue, company name, seller's name, contact person and phone number, description of purchased items, quantity, unit price, shipping method, date and other relevant information will be specified in the order. One copy of the order is then used in a three-way (explained later) match.

2.3.2 Receiving report

The receiving report is the company's documentation of the goods it has received. The receiving report may be in paper form or it may be an electronic record. The quantity and description of the goods that are mentioned in the receiving report should be compared with the company's order data.

Once someone receive the purchase order information, it must be compared with supplier's invoice. For this reason, the receiving message is the second one of three documents in three-way agreement.

2.3.3 Vendor Invoice

The supplier or vendor will send the invoice to the company that has received the goods or some services for the loan. After receiving the invoice or receipt, the customer refers to this invoice or receipt from the vendor. Each vendor's invoice is listed as payable bill for the reason of another processing. Once the invoice has been verified and approved, the final amount will be moved from debited account to the company's account and debited to another account (most often as an expense or property).

The most frequently used method for verifying the supplier's invoice is a three-way match.

2.3.4 Three-way match

The accounts payable process often uses a technique known as a three-way match to ensure that only valid and accurate invoices are recorded and paid. The three-way agreement then includes the following:

- Purchase order,
- Receiving report,

- Vendor Invoice.

If the data in all three documents agrees the supplier's invoice is entered into a payable account and scheduled for payment.

2.3.5 Vouchers

Some companies use a voucher to document or guarantee that the approval process is successfully completed. The voucher can be displayed as a cover sheet for attaching supporting documents (document order, message receipt, supplier's invoice, etc.) and for registering approvals, account numbers and other information for each invoice or account from the vendor.

After the payment of the supplier's invoice is successfully completed, the voucher and its attachments (including a copy of the check that was issued) will be stored in a paid voucher and / or invoice. Unpaid invoices and bills will be stored in an open file.

2.3.6 Vendor invoices without purchase orders or receiving reports

Not all vendor invoices have purchase orders or receiving reports, it means that three-way match agreement is not always possible. For example, the company does not issue a purchase order for a predetermined amount of electricity for the following month. The same works for telephone bills, natural gas bills, water bills, etc.

There are also payments that are required to be paid each month to meet rental contracts or other negotiated contracts. Between examples we can find monthly rent for a warehouse, office rental, car payments and so on. Even when these obligations will not have any order documents, responsibility remains unchanged, only legitimate and accurate amounts are paid.

2.3.7 Statements from vendors

Suppliers often send to their customers statements indicating amounts that have not been paid yet (indicated by the invoice number). After receiving the statement from the supplier, details of the report should be compared with the company records.

When company receives two same invoices and a vendor's declaration, means, that there is a possibility of the payment been duplicated. To avoid duplicate payments, companies often pay only for vendor invoices and at the same time they never pay for vendor statements.[12]

2.4 Accounts Payable - FI-AP modul

Accounts Payable Module alias SAP FI-AP records and manages accounting data for all company vendors. It is also an important part of the purchasing system where deliveries and invoices are managed respecting agreement made with the supplier. The system automatically performs accounting in response to operational transactions. Likewise, the system delivers Cash Management application, which returns detailed data of the invoice to optimize liquidity planning.

Payments are paid with the payment program. The payment program is implemented in FI-AP and supports all the standard payment methods (such as checks and transfers) in printed form and in electronic form (exchange of data media on disk and electronic

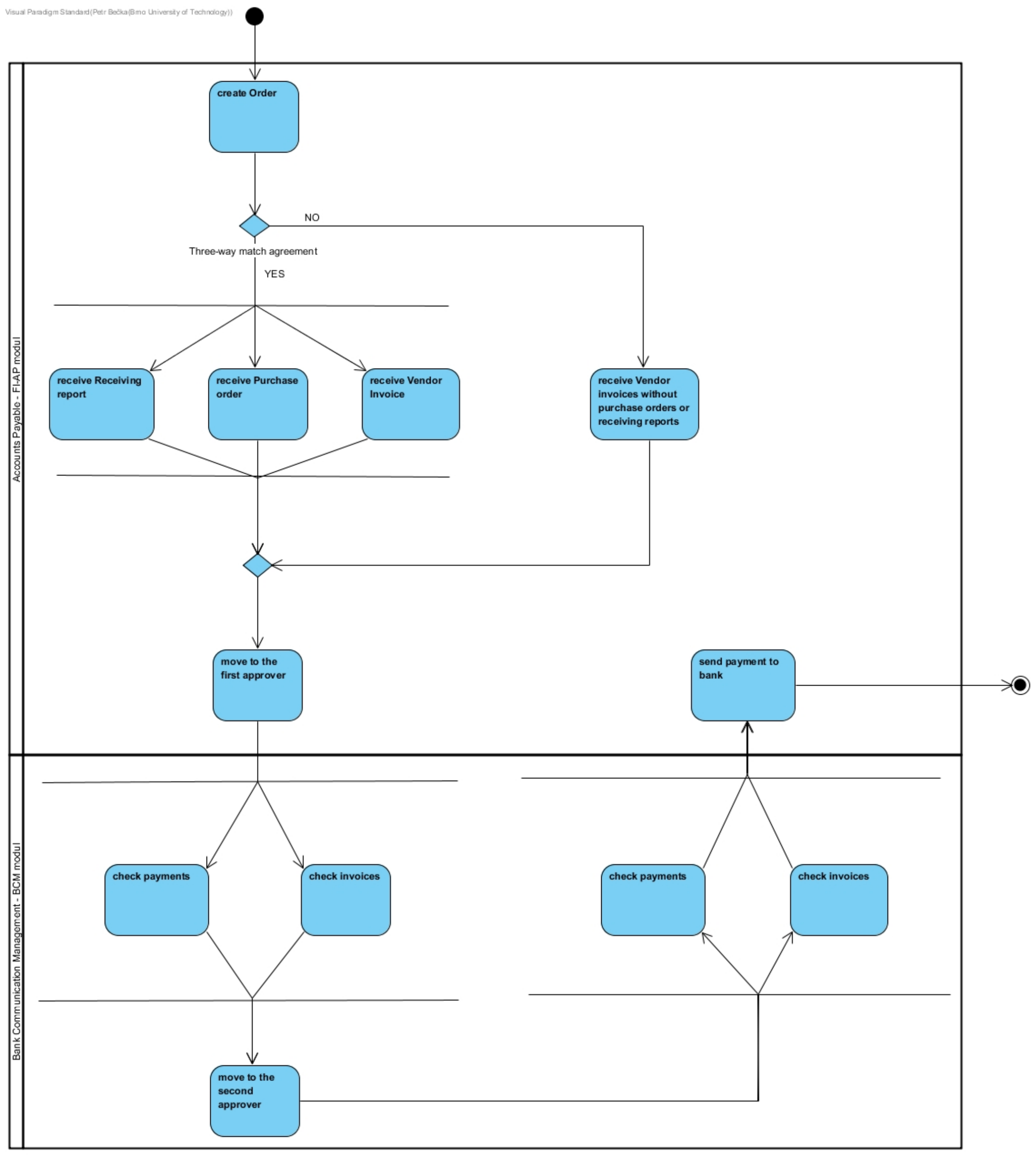

Visual Paradigm Standard(Petr Bečka(Brno University of Technology))
Accounts Payable - FI-AP modul
create Order
NO
Three-way match agreement
YES
receive Receiving report
receive Purchase order
receive Vendor Invoice
receive Vendor invoices without purchase orders or receiving reports
move to the first approver
send payment to bank
Bank Communication Management - BCM modul
check payments
check invoices
check payments
check invoices
move to the second approver

Figure 2.2: Diagram of activity showing work flow of processing created orders from creating purchase order, waiting for three documents necessary for a three-way match or processing payment without some. In BCM module is checked validity of all documents. If they are valid, FI-AP module mediates bank payment.

data exchange) as well. This program also covers country-specific payment methods. If necessary, the program supports the functionality for creating reminders to outstanding claims (like receiving a credit note).[24]

The debits made in the FI-AP are simultaneously recorded in the general ledger, where the different account types of the general ledger are updated on the basis of the realized transaction (for example commitments and advances). The system includes maturity forecasts and other standard overviews that can be used to monitor open items in payment accounts. FI-AP module configures balancing confirmations, account statements, and other notifications to meet the correspondence requirements. Balance sheets, balance records, and other evaluations are available to document transactions in payment accounts.[7]

2.5 Bank Communication Management - BCM modul

BCM, or Bank Communication Management is used to manage multiple banking interfaces. Allows to connect to the bank system, track the life cycle of created payment transactions, and improve direct payment processes. The module is also responsible for creating and approving batches that include individual payments, monitor the status of bank payments and their statements.[3]

With the Bank Communication Management module, payments in one or more payment flows can be presented and processed in the payment status monitor. The monitor is also responsible for transmitting status messages from SWIFT and individual banks. Individual payments could be split into one or more payment batches as needed. For example, you can group payments based on the amount to be paid - bigger amounts and smaller amounts. You can use a digital signature to approve, reject or defer the batch.

The SAP Bank Communication Management module is part of the SAP ERP Financial package. It allows to close integration of processing payments in ERP Financial.[4]

2.6 Approve Payments in SAP

The entire payment process is carried out in SAP using the Accounts Payable FI-AP module which is part of SAP S/4HANA solution. The workflow is modeled on figure 3.1 However, the process of approving and verifying individual payment batches is implemented in the BCM module. Without that, it would not be even possible to automate the confirmation of payments process. that would lead into inefficient work, especially in large companies.

When performing an action from a certain part of the payment process above the batch, the corresponding work item entry must be also automatically executed on the system. Depending on the release strategy (steps that need to be taken before the payments are released) defined for the batch (including payments), the approval of the first user may also be the final approval. In this case, the system will automatically create a payment medium for that particular batch to transfer into the bank. If more approvals (usually two approvers) are required, the system instead creates additional work items for the next approval step. These work items will be visible in other users' work lists. In the multi-user approval process, it is very important that none of the approvers can be present in the role of the other one. As a result, it would mean that one of the approvers will confirm the payment twice, and they could be paid without the second participant's notice.[1]

In conclusion it means that in approval process exists two kinds of approver:

- first approver,
- second approver.

2.6.1 Approving with one approver

As the first approver, the user can change the batch by removing specific payments from him. These payments can either be defered (so that they can later be included in another batch) or can be totally rejected. If this happens, the corresponding payment documents in the FI module should be canceled. The entire batch can also be defered or rejected as well. Finally, the approver can confirm the (remaining) batch and the system proceeds to create a payment medium.

2.6.2 Approving with two approvers

The same like the first approver, the second one can see the changed and released batches. In the role of the final or later approver there appears the next steps to release batches. In this role it is no longer possible to change the content of the processing payment batch, can either be approved or (if it should not go to the bank) be sent back to the first approver. All actions and changes are recorded and executed only after confirmation by the approver.

Chapter 3

SAP S/4HANA

SAP Business Suite 4 SAP HANA or SAP S/4HANA is a real-time digital solution for enterprise resource management that is developed on the SAP database operating system and SAP HANA platform. The goal of this product is to cover everyday business processes. Because it runs only on the SAP HANA database, it is offered as one product and integrates both enterprise and industry solutions. It offers also the SAP Fiori user interface, SAP S/4HANA can be run as a cloud solution or as an on-premise solution directly at the customer side.

This application will be developed and later on deployed on this platform. In spite of the fact that only the implementation of the user interface and database queries will be part of this work, it will also be necessary to connect this parts with the other SAP S/4HANA components to be able to run the application. (overview of components structure on figure 3.1) The data that will be retrieved from the database will be processed using logic located on a server that is implemented in ABAP. This existing layer also provides control of user authorizations and their rights to read and edit data. This logic is then available on a specific URL address, which is provided by SAP Gateway. Data exchange between the client side and SAP Gateway is then based on the OData protocol.

3.1 SAP HANA Database

SAP HANA represents an application and database platform that enables fast processing of large volumes of data in real time as well as processing of their instant analysis. SAP HANA is built to achieve fast response to database queries. One of the key features is the location of entire databases in RAM memory - this technique is called the in-memory database and because of that, compared to a regular SQL database, the processing time is much shorter. It means that the data is not stored primarily on hard disks. Hard drives are still used, but only passively. The standard process that allows database recovery is regular logs creation. The data are written into log files that are stored in persistent memory. This important process guarantee durability in enterprise applications. These properties ensure reliability of database transactions and are considered as basis for reliable enterprise computing technology.

Another important information is how data storing is implemented. HANA database primarily store data in columns rather than in rows. In practice it means that if the database contains, for example, data of the inhabitants of the Czech Republic, there are certain surnames that appear a lot more that others (like Novak), but the data are stored

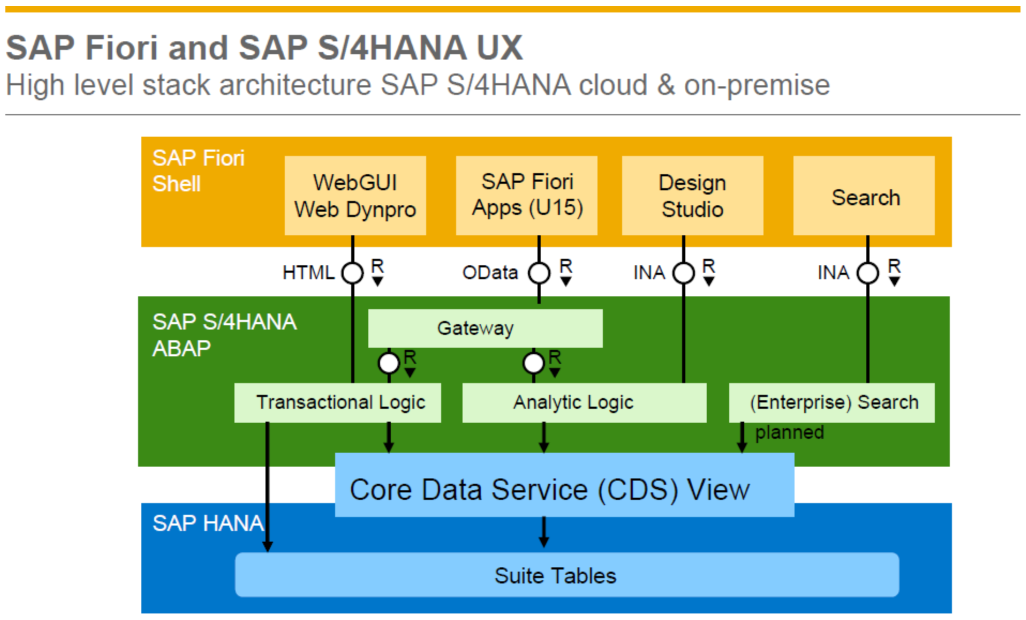


Figure 3.1: SAP S/4HANA architecture overview illustrating selecting data from database tables with Core Data Service views that is a part of the thesis. Then extracted data are modified in ABAP layer and this backend logic is then placed in SAP Gateway and provides data to user interface. Data are consumed by user interface, that is part of this thesis as well, by OData protocol. Source: [32].

in columns, the corresponding surname will only be saved in the column once and could be searched much faster.[30]

However, saving data into rows is also supported. SAP HANA also works as a application server with already built features with HTML5. Among other things, it supports development of new applications on mobile devices such as smartphones or tablets.

3.2 Core Data Services

Core Data Services (CDS) is an infrastructure that helps developers create a durable data model that transfers application services to client interface users. The developer can define analytic and data persistence models that are used to transmit data in response at the request of the HTTP client. CDS allows you to define the persistence model that contains objects such as views, spreadsheets, and structured types. These objects determine the data and how to use them in the application.[30] CDS as a data definition language is also extended by entities, associations, calculated fields and annotations that add semantic meaning to extracted data. (visualized on figure 3.2)

With CDS the development paradigm has shifted. The rule is that to get the best performance, it is necessary to move as many calculations as possible to the database system.

3.2.1 Entities

Core data services bring a framework for defining semantic data models on the central database of the application server based on the data definition language (DDL) and the data control language (DCL) managed by ABAP Dictionary. It means that CDS entity is defined as source code in the CDS data definition.[29]

3.2.2 Annotations

When an object defined in CDS source code is activated, the metadata defined by the annotations is saved in internal accesable database tables. This is how it works for every annotation with correct syntax. ABAP annotations and component annotations are evaluated in different ways:

ABAP annotations define technical and semantic attributes of a CDS object. They are usually evaluated for every CDS object when activated by the ABAP runtime environment. ABAP annotations can modify the behavior of Open SQL statement and provide access to a CDS entity. Value of an annotation is saved in special table with a translatable language key.

The component annotations are evaluated by the frameworks of the corresponding software components using an API. Names and values must follow the rules of the relevant framework. For SAP components, these can be taken from the tables of the SAP annotation documents.[25]

3.2.3 Associations

Associations are used to define relationships between entities, they are specified by an element added to a source entity with an association that points to a target entity, it is complemented by cardinality and by keys designed to join entities.

Target Entities in associations specify the target data holder that should be accessed represented by name of an entity in a CDS document. A target entity specification is requiered because there does not exist something like default target value in an association relationship.[26]

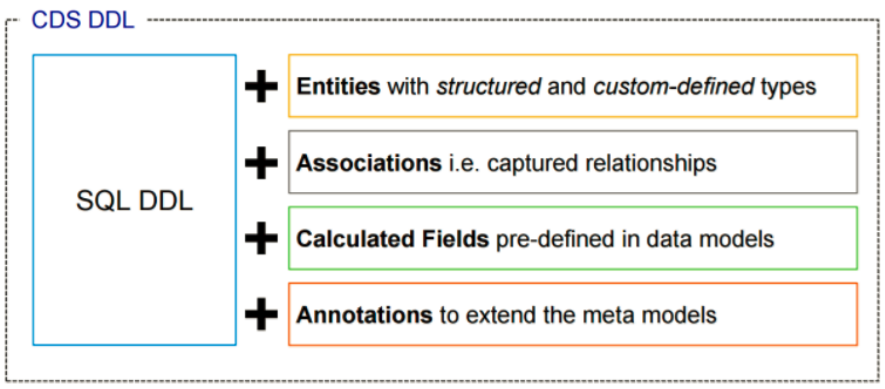

CDS DDL
SQL DDL
Entities with structured and custom-defined types
Associations i.e. captured relationships
Calculated Fields pre-defined in data models
Annotations to extend the meta models

Figure 3.2: Image showing Core Data Services composition. Source: [13]

3.3 SAP Gateway

SAP NetWeaver Gateway is a technology that provides a way to connect any device, environment, and platform to SAP software. The framework allows development of centralized solutions that bring social and collaborative environments, mobile and tablet devices and web applications into software.

It means that Netweaver Gateway is a set of ABAP add-ons to an existing SAP ERP system that provides access to business information for the user and reduces the complexity for consuming data that are exposed by Web Services. Gateway is an ABAP-based add-on and can be installed in three configurations. To develop new, or modify existing Web services is primarily ABAP programming language knowledge. Consuming these exposed Web services is reachable by making AJAX and network calls to specified URL. SAP Gateway handles secutrity issues by using existing system user authorizations, and provisions have been built into the product for SSO (single sign-on), CSRF (Cross-Script Request Forgery) and HTTPS.[11] No additional knowledge of the internal SAP operation is required. Gateway provides an API to access business data and features in SAP systems.

Consuming data from SAP NetWeaver Gateway require an HTTP(S) query creation capability. It does not require usage of any SAP software or protocols. Data consumption through SAP NetWeaver Gateway does not require knowledge of ABAP software implementation or even an understanding of SAP's internal system.

Using well known development tools such as Microsoft .Net, Apple XCode, or Open Source languages such as Ruby, PHP, or JavaScript, we can easily create user interfaces for business data by consuming OData (explained in next chapter) from the client side provided by SAP NetWeaver Gateway server.[17]

3.4 OData protocol

Open Data Protocol (OData) allows you to create REST-based data services that allow you to publish and edit web clients using simple HTTP messages specified by the URLs defined in the data model. This specification defines semantics and protocol behavior.

OData is an application-level protocol for interacting with created data via the RESTful API interface. The protocol supports description of data models, editing and querying data by these predefined models. It provides description for:

- metadata - machine readable description of a data model (figure 3.3),
- data - set of data entities and relationships between them,
- query - requires the service to perform a set of filters and other transformations to data and then returns the results,
- editing - creating, updating and deleting data,
- operation - invoking own logic,
- dictionaries - assign own semantic value.

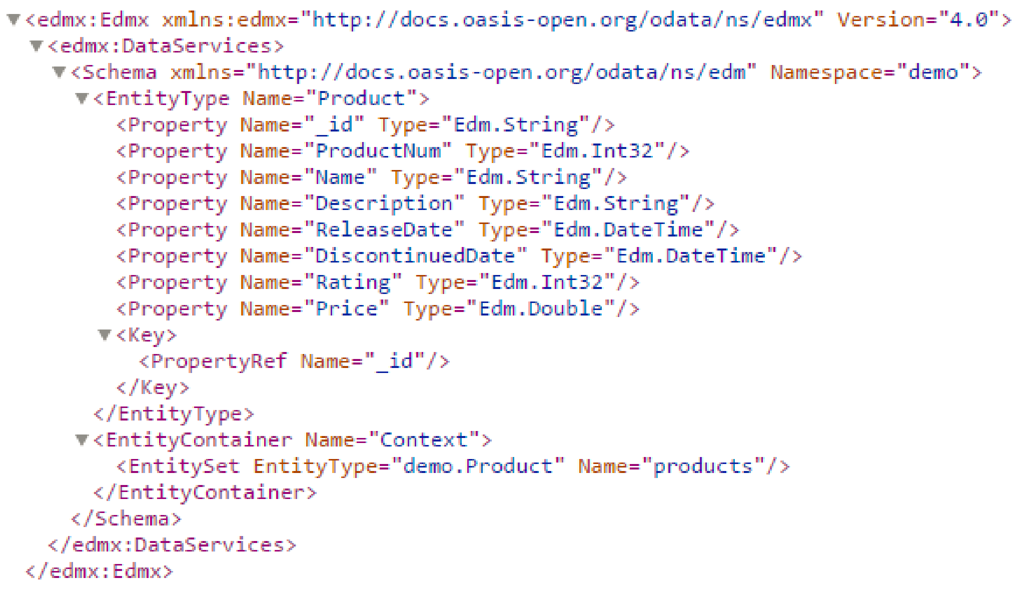
The OData protocol is different from other REST-based protocols because it provides a unified way of describing data and data models. This improves semantic interoperability between systems.[21] For this purpose, the OData protocol is guided by the following principles of proposal:

- preference of mechanisms that work on different data repositories
- not taking a relational data model,
- importance of extensibility - services should be able to support enhanced functionality without side effects on the client side of the application that is not informed about these extensions,
- following REST API rules,
- OData should be built step by step way - a very basic service should be easy to build,
- leaving it designed simple - only for common cases and possible extensions.

3.4.1 Data model

This section provides a description of an abstract data model that is used to describe OData protocol. OData metadata document is a representation of a service data model that is used for a client consumption. Central extended data model concept contain entities, relationships, sets of entities, activities and functions. Entities are in the end instances of entity types (like customer, employee, etc.).

Entity types are named with structured types with key. They define named properties and relationships. Entity types can be derived from inheritance of other already existing entity types. Key is made up from a subset of primitive properties (such as CustomerId, OrderId, LineId, etc.) of the entity type. Complex types are key less structured types consisting of a set of properties. These are types of values whose instances can not be referenced outside of their content. Complex types are commonly used as values of properties within an entity or like parameters for operations.



```
▼<edmx:Edmx xmlns:edmx="http://docs.oasis-open.org/odata/ns/edmx" Version="4.0">
  ▼<edmx:DataServices>
    ▼<Schema xmlns="http://docs.oasis-open.org/odata/ns/edm" Namespace="demo">
      ▼<EntityType Name="Product">
          <Property Name="_id" Type="Edm.String"/>
          <Property Name="ProductNum" Type="Edm.Int32"/>
          <Property Name="Name" Type="Edm.String"/>
          <Property Name="Description" Type="Edm.String"/>
          <Property Name="ReleaseDate" Type="Edm.DateTime"/>
          <Property Name="DiscontinuedDate" Type="Edm.DateTime"/>
          <Property Name="Rating" Type="Edm.Int32"/>
          <Property Name="Price" Type="Edm.Double"/>
        ▼<Key>
            <PropertyRef Name="_id"/>
          </Key>
        </EntityType>
      ▼<EntityContainer Name="Context">
          <EntitySet EntityType="demo.Product" Name="products"/>
        </EntityContainer>
      </Schema>
    </edmx:DataServices>
  </edmx:Edmx>
```

Figure 3.3: Example of the structure of OData metada file. Source: [23]

Relationships from one entity to another are represented as navigation properties. Navigation properties are generally defined as part of an entity type, but they can also be displayed on entity instances as undeclared dynamic navigation properties. Every relationship has its cardinality.

Enumeration types are the naming for primitive types whose values are constants with basic integer values.

Type definitions are primitive types with fixed values such as maximum length or accuracy. Type definitions can be used instead of primitive specified properties such as property definitions.

Sets of entities are named collections of entities (for example, Customers are a set of entities that contain customer subjects). The key in the set identifies an entity in an entity set. If multiple entity units use the same entity type, the same combination of key values may appear in more than one entity group and identify different entities, one per unit entity on which this key combination is displayed. Each of these entities has different entity id. Entity sets provide entry points into the data model.[22]

3.5 SAP Fiori

SAP Fiori is brand new user interface for SAP applications. It provides a platform of applications that are used on daily basis in business processes. There are included financial applications, computing applications, approval applications, etc.

Technology delivers real-time solutions for all existing business roles available for portable devices as well. For each role, it provides easy to use applications that are adaptable to smartphones, tablets, desktops and / or laptops.

Based on the survey, it was found that customers are using applications that focus on interaction between the employer and the employee, for example approving business trips and vacations. These applications had more than 300,000 screens with a variety of functionality. After identifying the most visited transactions, features, it has been decided that the existing solutions need to be redesigned and simplified. That's how SAP Fiori development has started.[8]

Application development is directed by five rules. By helping to regulate development into easy-to-understand applications and easy to control. These rules are:

- role separation - SAP has divided individual transactions and created user-friendly applications that display data based on roles in the system,
- reactivity - if SAP Fiori is combined with a strong SAP HANA database, as a result we get short response time for a database query,
- simplicity - to create the best user experience, it was necessary to keep the application simple, so development follows the rule 1-1-3 (1 user, 1 use case, maximum of 3 screens),
- availability - SAP provides all Fiori applications based on the same language, development platform does not matter,
- usability - SAP Fiori is designed to work with ECC 6.0 to make it easy for users to develop into an existing SAP system.

SAP Fiori Launchpad is the root element of all Fiori applications, and users can access individual applications via tiles. The tile is therefore the basic navigation element in Fiori Lauchpad, which is built as flexible and customizable item.[9] At Fiori we can find exactly three types of applications:

- transactional applications - usually represent views and interactions with existing business processes and solutions,
- analytical applications - typical representatives are the so-called Smart Business (KPI Cockpits) applications for analyzing the business plan implementation, but also other analytical, predictive and planning applications,
- fact sheet application - a search application that supports a combination of different applications so users can navigate from data view screen to a data editing action screen.

3.5.1 SAP Fiori Launchpad

The SAP Fiori Launchpad is the entry point of Fiori technology for both mobile and desktop devices. It is a page that basically works as a container for all applications that are available for the logged-in user in the system. Through all applications, it provides personalization, navigation, global filter settings, etc. Example on figure 3.4. Each tile in Fiori Launchpad is an application that can be run by the user. SAP Fiori tiles run on multiple types of devices and provide a single access point for business applications provided by SAP, these are transactional, analytical, intelligent, and intelligent business applications.

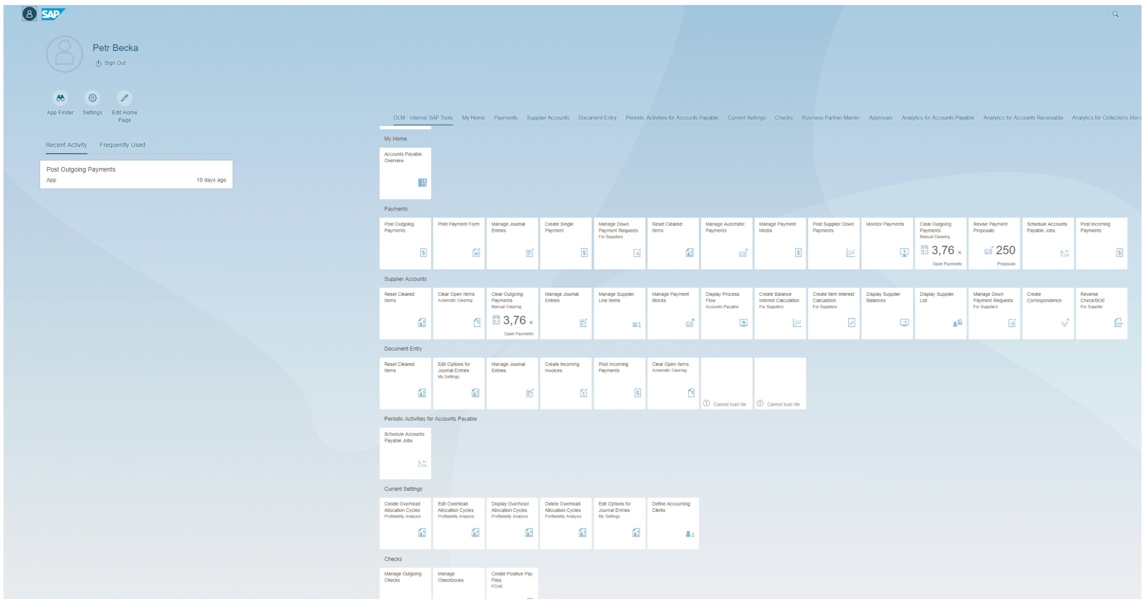


Figure 3.4: Fiori Launchpad example showing available tiles (applications entry point) and user menu in the left side panel.

3.5.2 SAP Web IDE

SAP Web IDE is a extensible, web-based development tool built to simplify application development. It provides templates, samples, code and graphical editors and modelers to speed up application development.

SAP Web IDE is designed to be easily used to launch new business scenarios when developing new application. With graphical extension editor it is easy to add some predefined extension point into new and existing applications as well. For mobile development there is integrated a ready to use tool for developing mobile hybrid applications called Cordova.[28]

Using SAP Web IDE user can also discover a try built in services for developing Big Data analytic applications and IoT (internet of things) scenarios. For SAP HANA IDE provides web-based set of tools for creating native SAP HANA applications. User is able to develop database models, calculation views, stored procedures, business logic and more.

Chapter 4

Front-end technologies

The purpose of this chapter is to get an overview of the most commonly used and best known JavaScript frameworks for creating graphical user interface. Consider each other and all their strengths and weaknesses.

4.1 ReactJS

React is a user interface library developed by Facebook, simplifying the creation of interactive and reusable user interface elements. Framework is commonly used and both Facebook and Instagram are written entirely in React.

The difference from other JavaScript frameworks is that this is a library for composing user interfaces, so it is not MVC framework. (could be seen on figure 4.2) It supports the creation of repeatable user interface elements that represent data that changes over time.

React does not use templates. Traditionally, the web application's user interface is built using HTML templates. These templates determine the full range of abstractions that can be used to create a user interface. React uses a real, full-featured programming language for building UI, which is an advantage for several reasons:

- javascript is a flexible, powerful programming language with the ability to create abstractions and that is very important for larger applications,
- by grouping a tag with the corresponding display logic, React can actually extend and maintain user interface,
- Creating tags and content in javascript means that there is a smaller vulnerability area for XSS.

JSX is an optional syntax extension to convert HTML code to raw JavaScript. React is very effective if the data changes at runtime. Traditionally, one has to look at what data has changed and imperatively change corresponding DOM elements to be up to date. Since React uses a fake DOM rather than a real one as you can see from figure 4.1, there is a special possibility to plot a fake DOM on the server - which means creating a server-side user interface.[31]

Once React component is initialized for the first time, the rendering method is invoked so creating the user interface is easy and fast. From displayed user interface is created a string of tags and inserted into the document. When the data changes, the rendering method is retrieved so that updates can be performed as efficiently as possible. Each component has

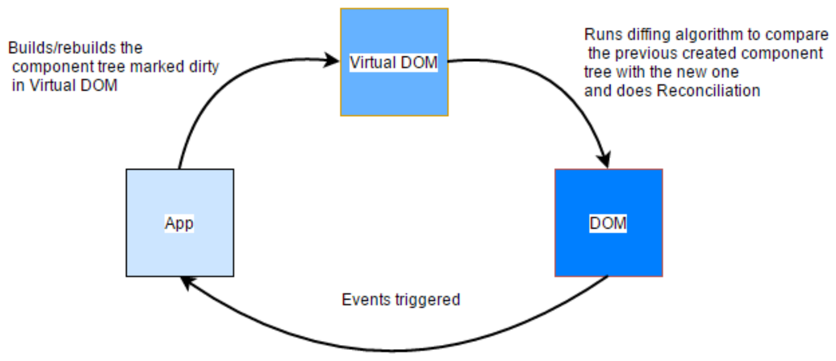

Builds/rebuilds the component tree marked dirty in Virtual DOM
Virtual DOM
Runs diffing algorithm to compare the previous created component tree with the new one and does Reconciliation
App
DOM
Events triggered

Figure 4.1: Relationship between application, DOM and virtual DOM created by React framework. Source: [19]

a state object and a props object, and the components can also set the default status of a component before any action is triggered.[15]

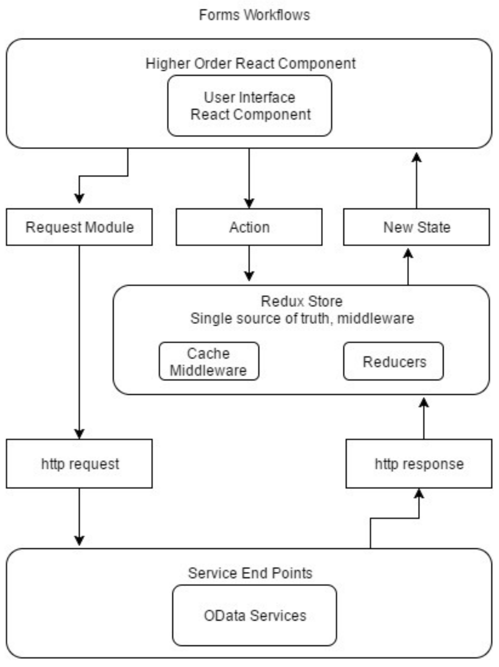


Figure 4.2: Basic architecture of a web application written in React framework. Source: [33]

4.2 Vue.js

Vue is a framework that is also designed to create user interfaces. It is designed to be able to adapt and gradually build on the required functionality. Core library contains only the display layer and is easy to integrate it with other libraries or existing projects.[5]

The composite system is very important for creating complex and difficult applications with reusable components. It is easy to split applications into more logical, independent parts.

Very important is huge support of the Vue framework reactivity. The individual components of the user interface implemented as JavaScript objects. This means that when you change an object, it also updates the component that represents the object's content. By passing a simple object to a Vue instance, it goes through all of its properties and converts them into the getter and setter (visible on figure 4.3) functions using the Object.defineProperty method. Getters and setters are invisible to the user, but meanwhile allowing Vue to track dependencies and changes when it's properties are accessed or modified.[6]

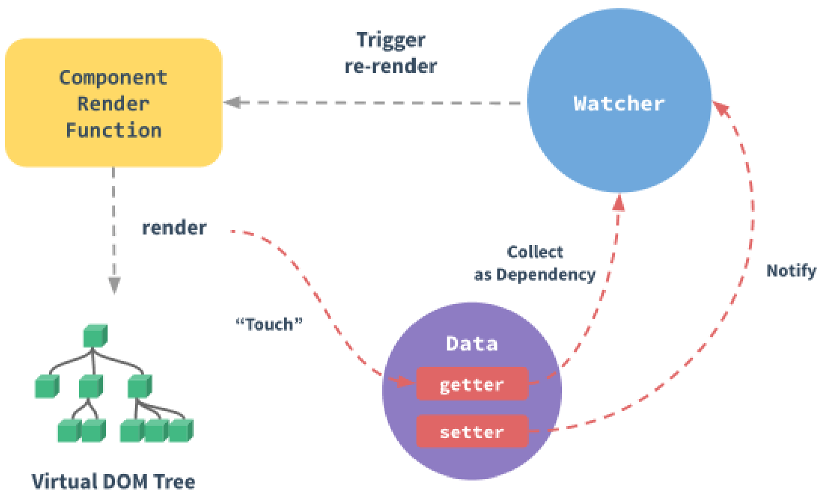


Figure 4.3: Workflow of Vue.js framework when rendering or changing displayed data. Source: [6]

4.3 Angular 2

The Angular Framework is used for creating user interfaces using HTML, JavaScript and / or TypeScript (which must then be recompiled into JavaScript). It is a compositional framework based on a set of libraries, which can optionally be extended by the import of other libraries. Source: [10]

Applications are created by composing HTML templates with Angularized tags, class components that control HTML templates. Application logic is contained in service classes and modules. The root module is always the first one to load.

All created applications are modular and Angular implements their own modular system called NgModules. Each application contains at least one NgModule which is AppModule root module. Modules are otherwise usually divided by application domain, workflow and / or related set of functionality as it is presented on figure 4.4 .

Modules are represented as classes containing decorator. Decorator contains a feature that modifies JavaScript objects based on metadata. As a result, this means that by assigning object metadata we know what it is used for and how it works.

Components work as a tool to control the displayed components. The application logic of each component is then defined here. Created classes interact with components using Angular methods and internal variables.[2]

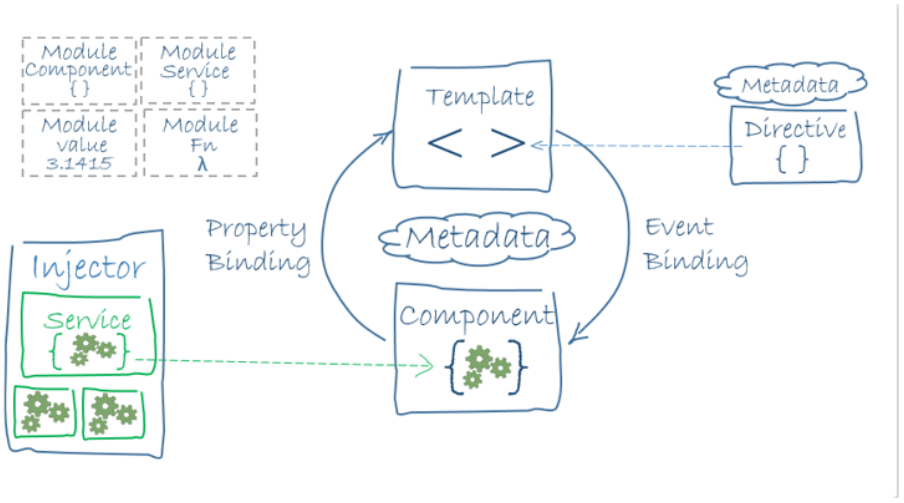


Figure 4.4: Image illustrating relationship between components and templates created by provided metadata description a provided service. Source: [2]

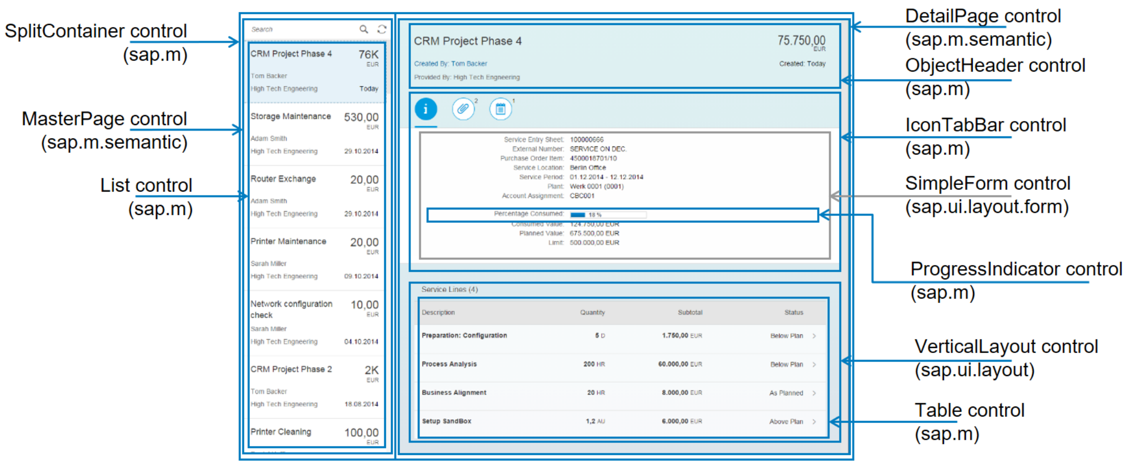
4.4 SAPUI5

SAPUI5 is library originated for rendering a client-side user interface. It contains a large number of predefined controls that build the final application. Another great advantage is the ability of bind data models directly to user interface controls.

Applications are created by folding individual SAPUI5 libraries, which are then combined into one application, each library contain different interface controls and allows us to access these controls functionality. Figure 4.5 shows screen combined from different controls from different libraries. SAPUI5 uses the Model View Controller concept to separate visualization of data from application logic and data model definition.

User Interface control is an object that defines the style and behavior of a given area on the screen. Individual controls inherit either the main control class sap.ui.core.Control or from other control to inherit or extend their basic functionality. They typically have some properties such as „text“ or „width“ that modify the way they are rendered on client-side. One control can also group other user interface controls. This means that it could behave like a container or user interface control. If the application can add child controls or a composite control, then the user interface control itself determines what are the child controls of the user interface and only uses these components.

View (user interface) is built up using the controls that are available in SAPUI5 libraries. Here is one of the biggest differences with other front-end technologies for creating user


SplitContainer control (sap.m)
MasterPage control (sap.m.semantic)
List control (sap.m)
DetailPage control (sap.m.semantic)
ObjectHeader control (sap.m)
IconTabBar control (sap.m)
SimpleForm control (sap.ui.layout.form)
ProgressIndicator control (sap.m)
VerticalLayout control (sap.ui.layout)
Table control (sap.m)
Search

CRM Project Phase 4 76K EUR
Tom Backer
High Tech Engneering Today

Storage Maintenance 530,00 EUR
Adam Smith
High Tech Engneering 29.10.2014

Router Exchange 20,00 EUR
Adam Smith
High Tech Engneering 29.10.2014

Printer Maintenance 20,00 EUR
Sarah Miller
High Tech Engneering 09.10.2014

Network configuration check 10,00 EUR
Sarah Miller
High Tech Engneering 04.10.2014

CRM Project Phase 2 2K EUR
Tom Backer
High Tech Engneering 18.08.2014

Printer Cleaning 100,00 EUR

CRM Project Phase 4 75.750,00 EUR
Created By: Tom Backer Created: Today
Provided By: High Tech Engneering

Service Entry Sheet:	100000666
External Number:	SERVICE ON DEC.
Purchase Order Item:	4500018701/10
Service Location:	Berlin Office
Service Period:	01.12.2014 - 12.12.2014
Plant:	Werk 0001 (0001)
Account Assignment:	CBC001
Percentage Consumed:	18 %
Consumed Value:	124.750,00 EUR
Planned Value:	675.500,00 EUR
Limit:	500.000,00 EUR

Service Lines (4)

Description	Quantity	Subtotal	Status
Preparation: Configuration	5 D	1.750,00 EUR	Below Plan
Process Analysis	200 HR	60.000,00 EUR	Below Plan
Business Alignment	20 HR	8.000,00 EUR	As Planned
Setup SandBox	1,2 AU	6.000,00 EUR	Above Plan

Figure 4.5: An example of a view built from the individual controls that are occupied in the libraries mentioned in brackets.

interface. SAPUI5 provides already completed functional elements such as tables, lists, inputs, filters, combo boxes, etc. The library then takes care of the rendering of each individual control and also provides the appropriate functionality.

The model stores the data and provides methods for loading these stored data. It also allows the presentation, automatic update and synchronization of data by linking view controls with data models. SAPUI5 supports several different data sources, such as XML, JSON and OData when linking is independent of the source type. A client-side model, most of the time JSONModel, is used to link user interface controls to JavaScript objects. JSONModel is primarily designed for small data sets. A serialized JSON string is typically used for data transmission. The second type of model is a server-side model, which is designed to retrieve data from server. It includes implementation of methods (POST, GET, etc.) that can be used to query for individual entities that hold data provided by SAP gateway service or for entities to save user-edited data.

Controllers are designed to handle events that are triggered by user, obtain user interface logic, communicate with the server and to create and store client-side and server-side models.

4.5 Evaluation

As mentioned above, React, Angular 2, and SAPUI5 stand for large companies that are actively developing, updating, and maintaining them. All frameworks are also based on components. This allows to pass the input into the component, after completing the internal functions, it returns the rendered part of the user interface, whose shape is derived from the input. React and Vue.js frameworks excel in creating so-called dump components that do not keep the internal state. Angular 2 allows to store the state of data inside the components. In any case, it is necessary to implement the rendering functions of user interface and the style of the displayed component. Here is also the advantage of the SAPUI5 framework that provides predefined components, including rendering functions and implementation of internal state preservation.

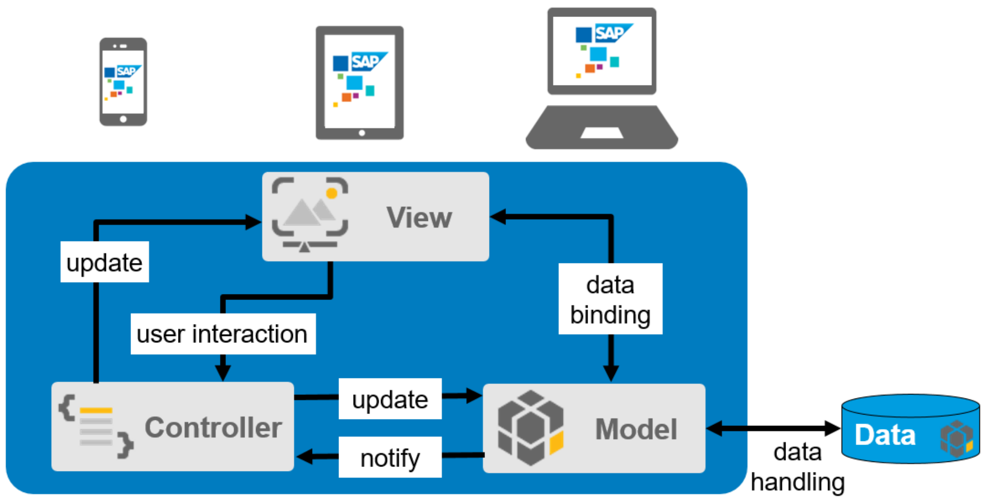


Figure 4.6: This is how MVC architecture works in SAPUI5 framework. Controller updates view when data is changed and handles user interaction. Model works like container full of data and moves with them between view and controller.

Another important aspect is ODATA consumption from SAP Gateway. Could be seen on figure 3.1. While with the React framework, you can use the npm package service to install the OData extension, for Angular 2 it is necessary to implement this functionality separately. SAPUI5 is natively ready to work with this protocol and provides methods for reading, deleting, and updating data over this protocol.

The biggest factor in choosing a framework was that SAP business applications are being deployed using the SAPUI5 framework, which means you can get inspiration when choosing the right components and working with data. It means that selected framework for creating user interface is SAPUI5.

Chapter 5

Application for approving bank payments

In the very first part of this chapter will be introduced how Fiori web applications are driven by the design while its development (figure 5.1). The stages of development, their meaning and content will be described. In the second part, a specific design for Approve Bank Payments application will be introduced and described. Chapter will then introduce the development of the application itself and its specific parts.

5.1 Methodology of development

This section describes and explains the methodology by which cloud customer applications are created in SAP environment. The methodology is called the Design-Led Development Process. With a strong focus on users and their needs, the process of developing and supporting developer teams is governed to deliver the best possible applications. The process uses proven ways of thinking to achieve optimal user experience. It's simple and easy to follow, providing a solid foundation for the scale design for approving applications as a whole.

Design-based development is useful because it supports unity during development between designers and developers while ensuring that end-user needs are addressed at every design and / or development step.

Design-Led Development Process covers the entire development life cycle of a web application and consists of three main phases and their corresponding layers. Of course, iterations are part of this approach and are strictly adhered to. If there is such a significant change that should be incorporated into the application and re-submitted to the design gates, then it is up to the development team to decide whether the changes will be incorporated.[27]

The individual phases are sequentially numbered and are called design gate 0, design gate 1 and gate design 2 (shorthanded: D-Gate 0, D-Gate 1, D-Gate 2). Phases have a predetermined fixed execution order, and summarize application development, from request generation, user role definition, UI design to custom implementation and testing. The section will illustrate clearly which of the three phases each of these activities belongs to. It will be said what is the input and output of each such part of the development and what steps are needed during this part.

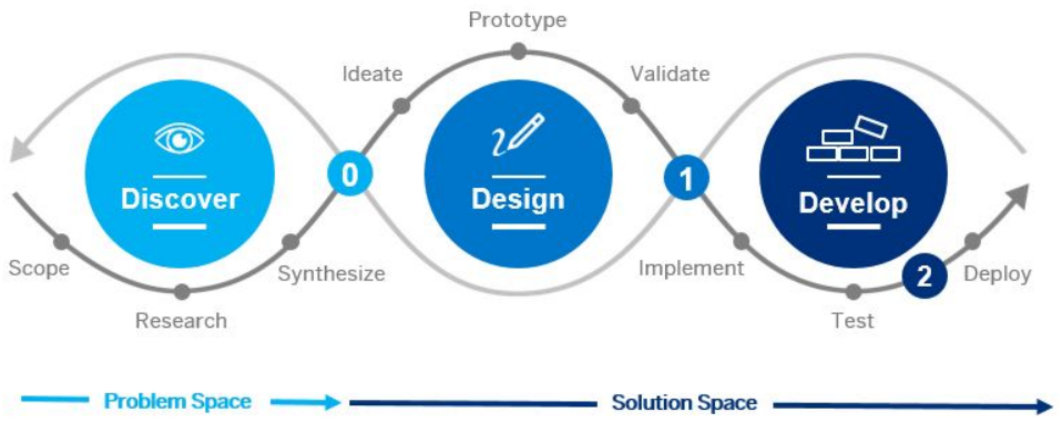


Figure 5.1: Picture representing process of development application from phase discover which ends with D-Gate 0, over to design phase that includes brainstorming and produces prototypes and ends with D-Gate 1. Up to development phase where application is built and tested and ends with D-Gate 2. Source: [27].

5.1.1 D-Gate 0

The goal of D-Gate 0, the discovery phase, is to understand the end user's business requirements and capture his problems and needs. Before diving into design and implementation, it is important to perform research with potential end-users and keep mockup of the application in the design area.

After gaining a clear understanding from the business point of view, collecting descriptions of typical representatives for the roles that appears in application and tasks to be solved, the team is able to leave this part and move on to the implementation part. An overview of the collected information and documents in Design Gate 0 are designed to determine whether the application is ready for design. A review in Design Gate 0 should be done before prototypes begin to be created.

Design Gate 0 ensures that user research is established and discussed with the customer, that developer has an overview of business roles and that application focuses on the end user's requirements.

To evaluate the completeness of Design Gate 0, the following facts help:

- business role description: an explanation of the business role, including industrial or business processes, from a commercial point of view,
- mapping application to business roles: explanation of context and content for a given issue, scheduled response and device-specific integration features,
- description of the end user: indication of all names of customers, regions and sectors involved in the research activities. Including the numbers and types of research activities and the number of end-users involved,
- personality: describe a person for this application, including details such as goals, job responsibilities, needs, competencies and obstacles in their everyday work,

- usage of use case diagram: description of user interaction with the system using case diagram for each relevant case of use.

Then there are also optional steps to help improve D-Gate 0:

- description of business processes: their high level description, showing where and how applications fit into business processes,
- observing a sequence of activities: explaining or drawing steps in the flow of user activity. Inclusion of all steps, needs and demands of users, work events and edge cases within the work.

5.1.2 D-Gate 1

In D-Gate 1, the design phase, the main goal is to create an initial design based on D-Gate 0. Here, it is important to really apply the principles of design thinking, develop design prototypes and decide what the technology should be used. Ideally, a team of scientists and developers should be created to brainstorm, create a scenario, and make a first prototype that can be verified by users.

Design-Gate 1 ensures the compliance of the planned design with the SAP Fiori Design Guidelines to create a user-friendly environment.

To evaluate the outcome of Design Gate 1, these artifacts will helpful to summarize the work and develop a design for each application:

- Fiori App Portfolio in OAM: create a record in the Fiori application portfolio in OAM to create a new application and deliver or extended version of existing applications,
- point of viewpoint: creating a view that includes user needs and insights based on user research and interviews. Understanding why user needs to perform certain activities,
- application basis: explanation of application. Using a comprehensible language, that can also be understood by people outside of the expertise needed, for application development. Unifying application before its designing,
- mocku-ps and prototypes: using simple mock-ups to help communicate ideas about the application and support design discussion, provide interactive prototypes to provide real-life experience with the future product.

5.1.3 D-Gate 2

The goal of the development phase, D-Gate 2, is to design and create a described product that has been requested and approved by the customer.

During implementation, it is often necessary to change some aspects of web application design for possible integration of user feedback, application optimization, or technical constraints. When an application is first implemented and tested, the initial design may require further iterations and modifications before the developer team arrives at the final design and implementation.

The purpose of Design Gate 2 is to ensure that the final implementation and design is done according to the SAP Fiori Design Guidelines. An overview is made using a demo version of the application that publishers prepared for reviewers.

5.2 Requirements analysis

This section corresponds to the D-Gate 0 phase of the Design-Led Development Process. There will be briefly described the requirements for the application, how the requirements were created and how they were specified. This first part of creation of the application should decide whether the application provides a new functionality that will bring new opportunities to the customer and simplify his work in some way.

To do this, of course, it is very important to have an experienced product owner who comes with an application idea. He must have a rich experience and a deep knowledge of the application domain. He also becomes the main person who is consulted about adhering to the application domain rules while development.

An important role at this stage is also played by a customer who works very closely with development team and discusses the functionality of the application. It is then, to a certain extent, tailored to the user.

This section will then mention the Approve Bank Payments design process, present requirements for the application, and the design of the specific functionality that should be provided and offered to end users.

5.2.1 Functionality design

As indicated on the use-case diagram in the figure 5.2, key functionality has been established for the resulting application to automate the process of approving bank payments for customers. First of all, the application was emphasized to provide functionality that covers the entire process of approving bank payments.

After discussions with customers, it was first determined and defined that the application would be made available for two user's roles (accounts payable manager and cash manager) because of the possibility of two way approval of bank payments and the prevention of fraudulent or hidden payments within companies during the process. This requirement follows from the general rules of two way bank payments approval process (see 2.6.2). Each user role will have very similar operations, but they will operate with different data based on assigned authorizations to access the particular data. For this reason, in the use case diagram of figure 5.2, both roles are modeled as a single user.

The basic functionality of the application is the payment batch manipulation. These batches contain individual payments that can be generated for one or more invoices. In the payment batch, then, on the dependency of a customer's environment set-up, payments that are somehow similar (depending on the settings already mentioned) are grouped together. The user must then be able to execute three types of operations over a payment batch. The first is the approval operation. If this operation is performed as a accounts payable manager role, it will cause the payment batch to be passed on to the user logged in as a cash manager role for further approval. If the operation is executed by the user logged in as a cash manager, the payment batch, including all payments, is approved and ready for further processing. In addition, the payment batch may be defered or postponed. This happens if, for example, payment cannot be paid at all for a variety of reasons or there is insufficient means to prove and check the validity of the payment batches. The last of three operations that could be done is a rejection. If a payment batch is rejected at any level of processing, then it is flagged as defected or incorrect and treated in accordance with the company's internal guidelines.

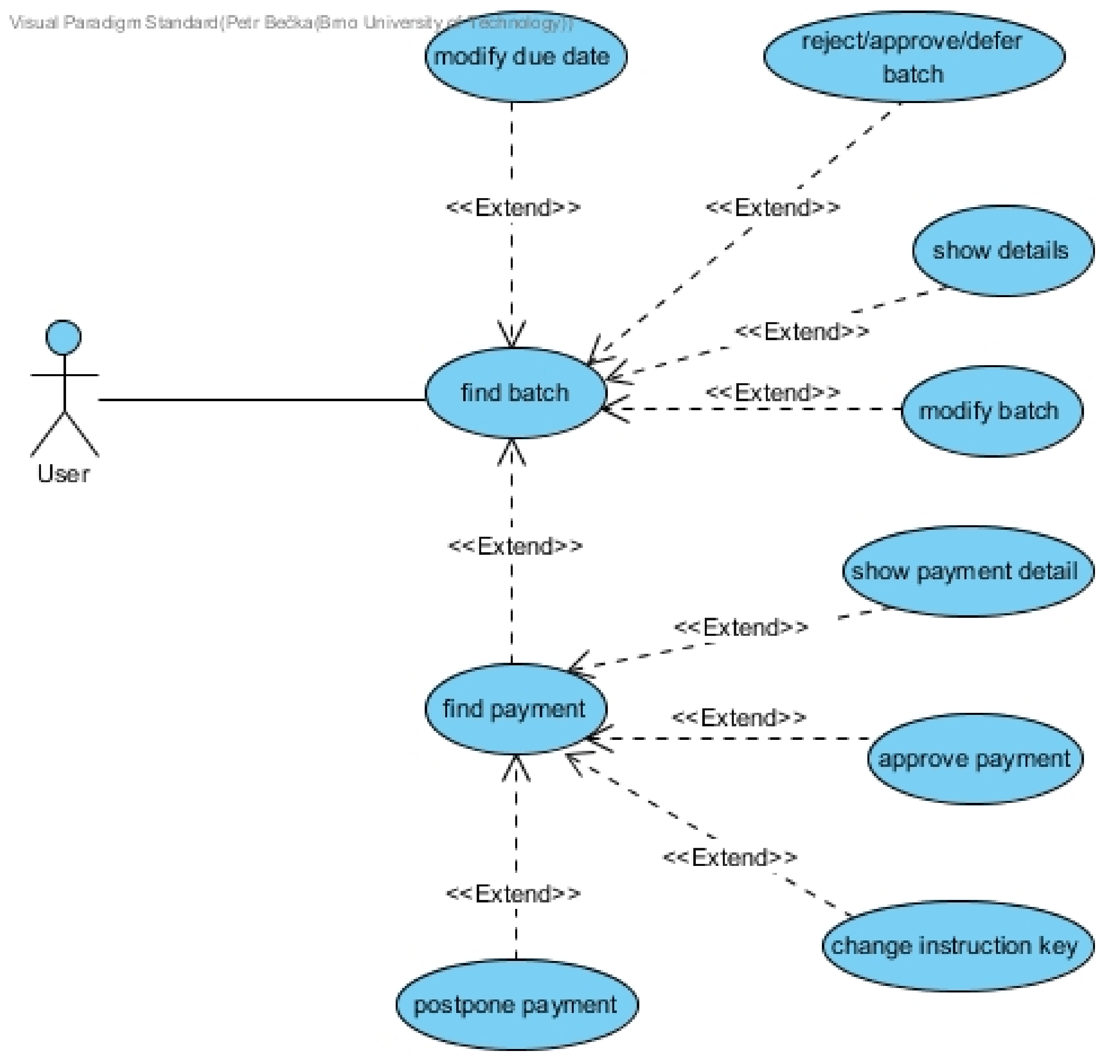


Figure 5.2: Use case diagram covering possible interaction between user and application.

Another key requirement for the application was a change of the due date value. This generally indicates the date by which the payment batch, as a whole, should be paid into the bank account. However, this date may be very depending on the conditions or changes in the contract, and it is therefore appropriate for the application to offer an opportunity to modify this value for a individual payment batch.

The payment batch must also include detailed information on which user is able to decide how payment process will be handled. Basic information to help identify the batch is the company code of the company to which the payment is sent, the total amount to be paid in local currency, the date of creation and the rules under which the payments are paid, called instruction key, are also important. In each organization or country, these rules are different and the rules are determined by the country or organization to which the payment is sent. Useful information may include the payee's account number, IBAN and house bank.

The application also provides the ability to modify attachments that are attached to the payment batch. There is visible the name of the user who uploaded the attachment and the date of the action. Last but not least, the user will be able to view individual payments from which the batch is composed, and then a timeline that expresses and accurately visualizes how the payment batch was treated in the past and who did particular changes.

The application was also designed for the case when only part of the payment batches need to be approved. It means that there is also possibility of approval at the level of individual payments. For this decision, it is necessary to display the important details that will allow such a decision, the details are, specifically for this application, payee name, amount, payment number, country key, instruction key and more.

For payments, user will also be able to view individual billing invoices. The sum of the individual invoice values then indicates the final payment amount. The company to be sent the payment could also change the way how bank payments are internally processed and deals over time. It is therefore possible to customize the payment key in an application with the instruction key field, which defines the process of repaying the liability to another party. It can, of course, also be rejected or postponed, as well as the whole payment batch.

5.3 Design of application UI

This chapter discusses the process of designing an application's user interface. From the perspective of Design-Led Development Process. This chapter is part of the D-Gate 1 phase. Its entry is the application requirements and the output is then a user interface design that is discussed with the product owner of the application and, of course, the customer. It will describe how the available data has been separated, and then using this separation, the individual screens, that can be made available to the user, are defined.

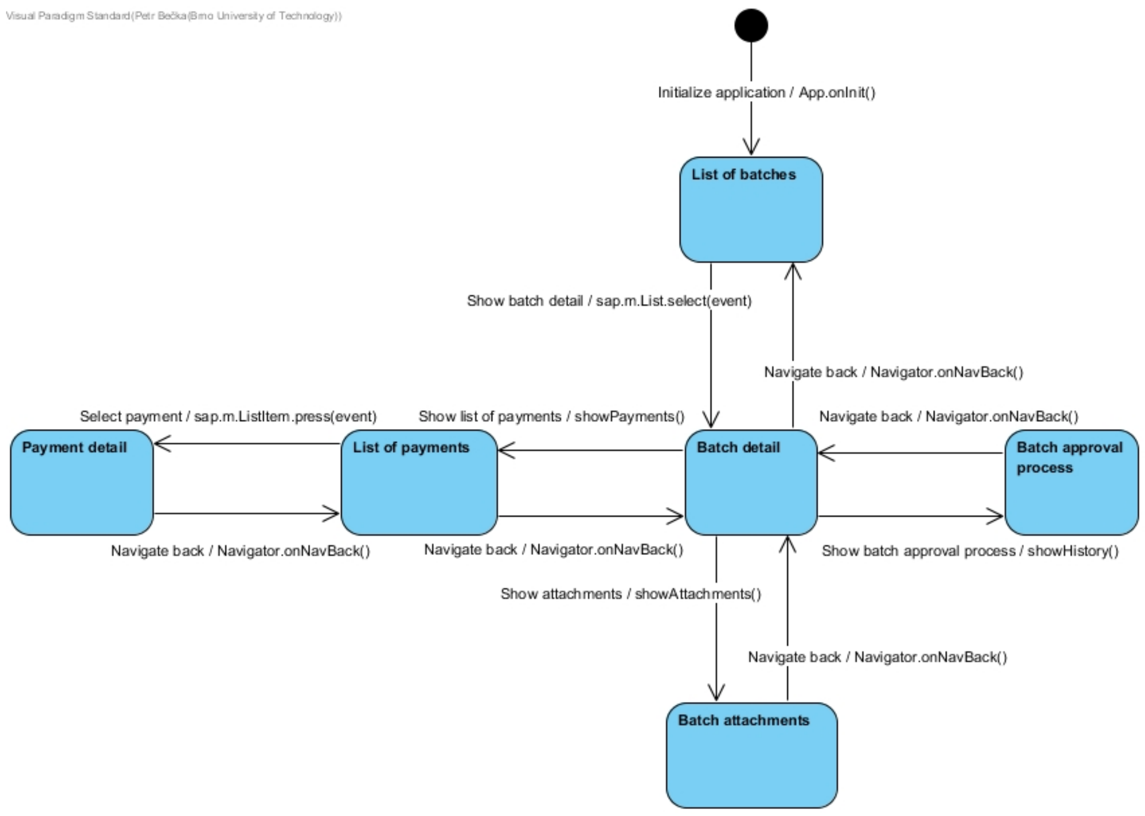
In the next phase, a detailed design of the application's user interface will be described and visualized as well as the UI components by which the data is transmitted and presented to the end user.

5.3.1 Design of data view

The data that will be displayed in the application is necessary to gather together based on their semantic meaning. This procedure will create several groups whose data are semantically very similar or describe a similar object. This section will introduce what data has been merged and how it has been grouped into individual groups. Each group at this moment expresses one screen that the end user can see. These screens are clearly shown in the picture 5.3.

It is crucial for users to recognize the payment batch they are currently looking for and over which they want to perform the appropriate operations. A screen containing a payment batches should serve for this purpose. Data that clearly defines a payment batch including the total amount, the company code of the company to be paid, the identification number, the creation date and the payment order (List of batches screen on 5.3) are visible there.

The next screen is the detailed information that can be obtained from the payment batch and which can play an important role in the approval of the payment batch. Here is also important to incorporate the detail payment batch screen option (Batch detail screen on 5.3).



Visual Paradigm Standard(Petr Bečka(Brno University of Technology))
Initialize application / App.onInit()
List of batches
Show batch detail / sap.m.List.select(event)
Navigate back / Navigator.onNavBack()
Select payment / sap.m.ListItem.press(event)
Show list of payments / showPayments()
Navigate back / Navigator.onNavBack()
Payment detail
List of payments
Batch detail
Batch approval process
Navigate back / Navigator.onNavBack()
Navigate back / Navigator.onNavBack()
Show batch approval process / showHistory()
Show attachments / showAttachments()
Navigate back / Navigator.onNavBack()
Batch attachments

Figure 5.3: Screen diagram about how the information will be passed on to the user.

A screen showing a list of payments that are linked to a payment batch (made up of these payments) could be another screen. It is important to consider the option that such payments could be many and the amount of information that could be displayed increases with each additional payment (List of payments screen on 5.3).

In addition, the screen for the history of actions, that were executed on the payment batch, was designed. Here is the name of the action that was taken, the date when this change was made, the user's note for the performed action that serves as additional information for following users (Batch approval process screen on 5.3).

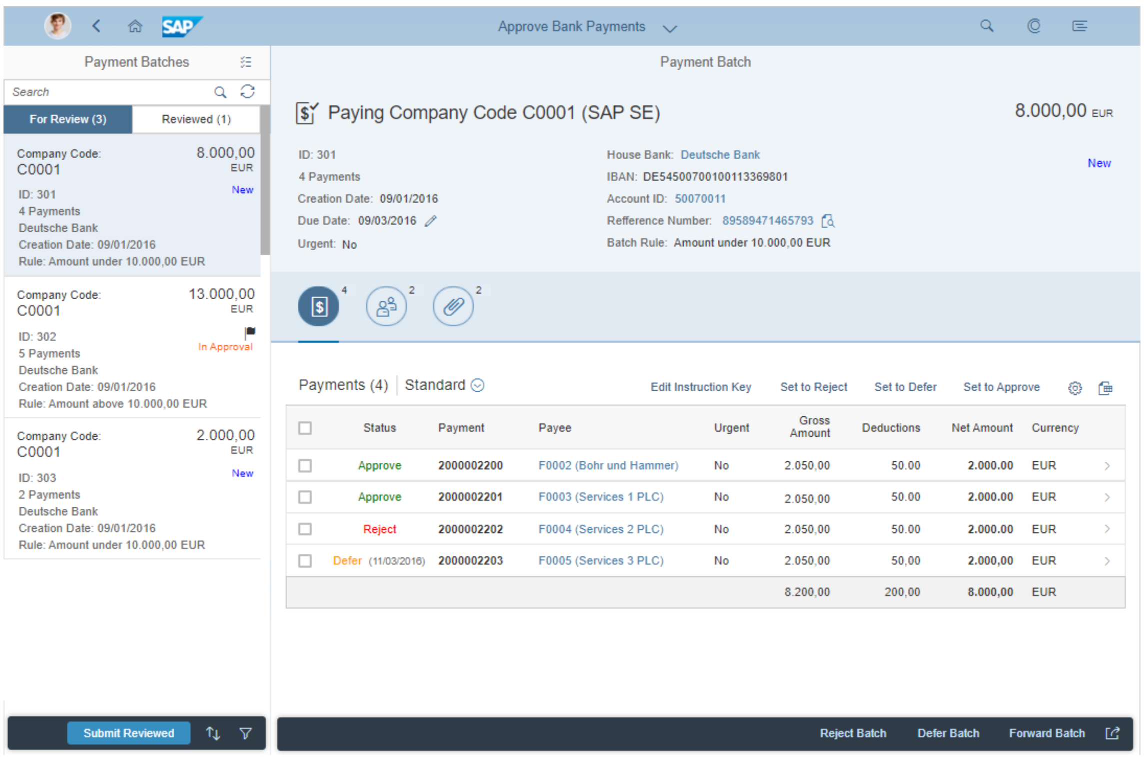
The next screen is an overview of attachments that is binded to the payment batch. There should be visible the name of the user who uploaded the attachment, the date when attachment was uploaded, the size and type of the file (Batch attachments screen on 5.3).

The final screen and a group of data that can be viewed separately from others are detailed information about the selected payment. This detail also includes a list of individual invoices and important data allowing user to decide whether approve payment or not (Payment detail screen on 5.3).

5.3.2 Design of data components

To maximize the user experience, the payment batches was firmly located on the left side of the application. Finding such a payment batch serves as the initial action of all other operations that can be performed inside the application, see use-case diagram 5.2. These batches will be displayed in a list that can be filtered and organized for an easier searching through the list.

The right part of the screen (visible on figure 5.4) is designed to be able to change its context. In the first case, it shows the detail of the payment batch at the top. This information will be displayed with text elements, or others when there is need to edit information. At the bottom of the right-hand part of the screen, the List of Payments, Batch Approval Process, and Batch Attachments screens (see figure 5.3) will appear based on the user's choice. From this, the List of Payments and Batch Attachments screens will be solved using a table that will set up columns that are important to users, sort individual records based on selected attributes, or filter through specific values for that column. The Batch Approval Process screen will be resolved using a timeline component to clearly visualize the actions taken and, in particular, the order in which the actions were performed.



Company Code: C0001 — 8.000,00 EUR — New
ID: 301
4 Payments
Deutsche Bank
Creation Date: 09/01/2016
Rule: Amount under 10.000,00 EUR

Company Code: C0001 — 13.000,00 EUR — In Approval
ID: 302
5 Payments
Deutsche Bank
Creation Date: 09/01/2016
Rule: Amount above 10.000,00 EUR

Company Code: C0001 — 2.000,00 EUR — New
ID: 303
2 Payments
Deutsche Bank
Creation Date: 09/01/2016
Rule: Amount under 10.000,00 EUR

Paying Company Code C0001 (SAP SE) — 8.000,00 EUR

ID: 301
4 Payments
Creation Date: 09/01/2016
Due Date: 09/03/2016
Urgent: No

House Bank: Deutsche Bank
IBAN: DE54500700100113369801
Account ID: 50070011
Refference Number: 89589471465793
Batch Rule: Amount under 10.000,00 EUR

New

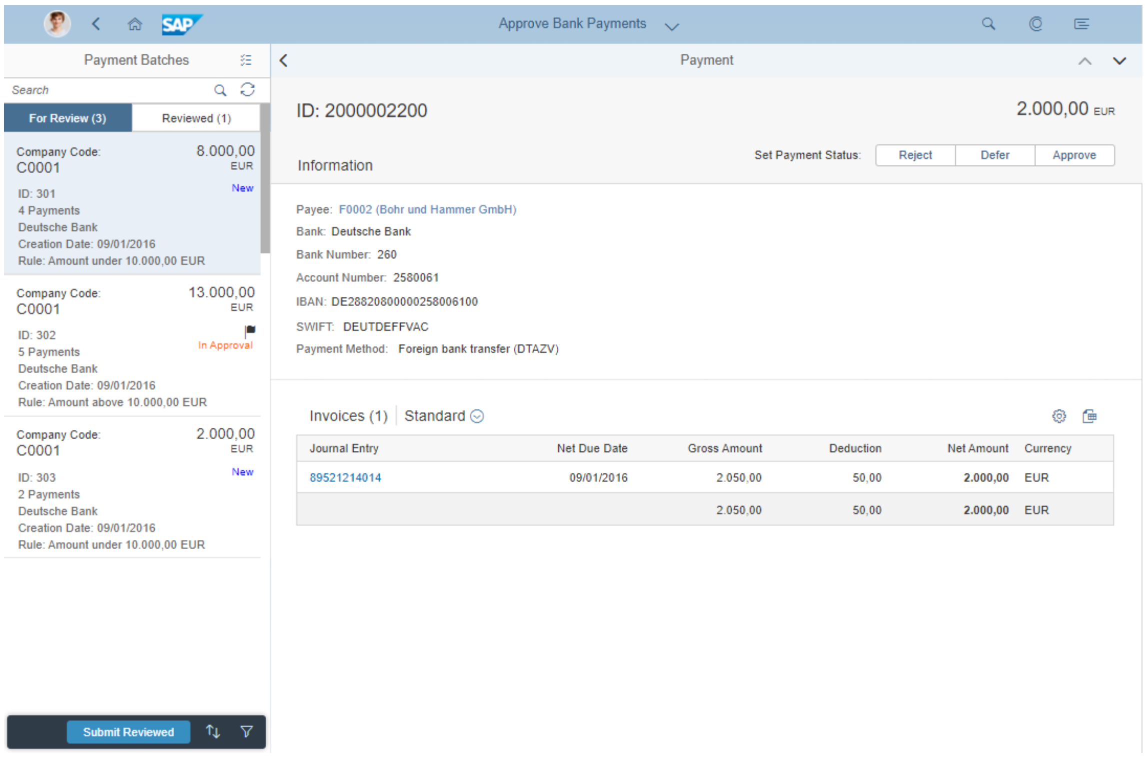
Payments (4) | Standard

Edit Instruction Key | Set to Reject | Set to Defer | Set to Approve

Status	Payment	Payee	Urgent	Gross Amount	Deductions	Net Amount	Currency
Approve	2000002200	F0002 (Bohr und Hammer)	No	2.050,00	50.00	2.000.00	EUR
Approve	2000002201	F0003 (Services 1 PLC)	No	2.050,00	50.00	2.000.00	EUR
Reject	2000002202	F0004 (Services 2 PLC)	No	2.050,00	50.00	2.000.00	EUR
Defer (11/03/2016)	2000002203	F0005 (Services 3 PLC)	No	2.050,00	50,00	2.000,00	EUR
				8.200,00	200,00	8.000,00	EUR

Figure 5.4: Mock-up of detail batch view provides detailed information about selected batch. There are, in the upper part, visualized some details inside text labels. The second part is showing table of batch payments and their status, timeline of payment approval history or attached documents.

In the another context (visible on figure 5.5), could be seen detailed payment information in the right part of the screen (Payment detail screen). This view can be navigated from the List of Payments screen. There will be on top, drawn as text elements, visualized detailed information identifying the payment itself. The lower part will then include a table component that shows the individual invoices that make up the displayed payment.



Company Code: C0001 8.000,00 EUR
ID: 301 New
4 Payments
Deutsche Bank
Creation Date: 09/01/2016
Rule: Amount under 10.000,00 EUR

Company Code: C0001 13.000,00 EUR
ID: 302 In Approval
5 Payments
Deutsche Bank
Creation Date: 09/01/2016
Rule: Amount above 10.000,00 EUR

Company Code: C0001 2.000,00 EUR
ID: 303 New
2 Payments
Deutsche Bank
Creation Date: 09/01/2016
Rule: Amount under 10.000,00 EUR

ID: 2000002200 2.000,00 EUR

Payee: F0002 (Bohr und Hammer GmbH)
Bank: Deutsche Bank
Bank Number: 260
Account Number: 2580061
IBAN: DE28820800000258006100
SWIFT: DEUTDEFFVAC
Payment Method: Foreign bank transfer (DTAZV)

Journal Entry	Net Due Date	Gross Amount	Deduction	Net Amount	Currency
89521214014	09/01/2016	2.050,00	50,00	**2.000,00**	EUR
		2.050,00	50,00	**2.000,00**	EUR

Figure 5.5: Mock-up of a detail payment page. Showing payment that belongs into selected batch from the list in the left side panel. User can defer, reject or approve payment and search through all invoices included in payment.

5.4 Application architecture

This chapter, called Application architecture, is designed to clarify and explain what the implementation of Fiori application takes. This is a description of the beginning of the D-Gate 2 phase of the Design-Led Development methodology used in SAP. First of all, there will be introduced what is necessary to be done for each Fiori application in general to be able to run such an application. It will indicate what parts it contains, what are the necessary files located, what they contain, the necessary settings and what techniques are used when implementing them. There will also be a solid link to the SAPUI5 javascript framework, which simplifies the whole process of creating an application itself, what advantages it brings for a developer. It will also indicate how each parts of the application work together and how the initialization and start of such applications are done.

In the second section, the specific architecture features of Approve Bank Payments application will be listed. These will be used and useful in the process of creating the application itself. In particular, we will deal with decomposition of application into logical parts, division of functionality into individual classes and adaptation of data models to individual application screens.

5.4.1 Architecture of Fiori application in general

Each Fiori application (figure 3.1 - SAP Fiori Apps) generally consists of several essential parts without which it could not be executable and could not run in general. Technically every application is registered as a component in the component container called Fiori Launchpad (section 3.5.1). Generally, if a user clicks on a tile representing a Fiori application, the Component.js file is uploaded using the index.html file and the Fiori Launchpad environment. The class implemented in this file then takes care of initializing the entire application. Since used SAPUI5 framework is based on MVC architecture, the required user view, the controllers that takes care of actions invoked by user on view, the data models through them the view and the controller move data, and the other customization files are also initialized (figure 5.6).

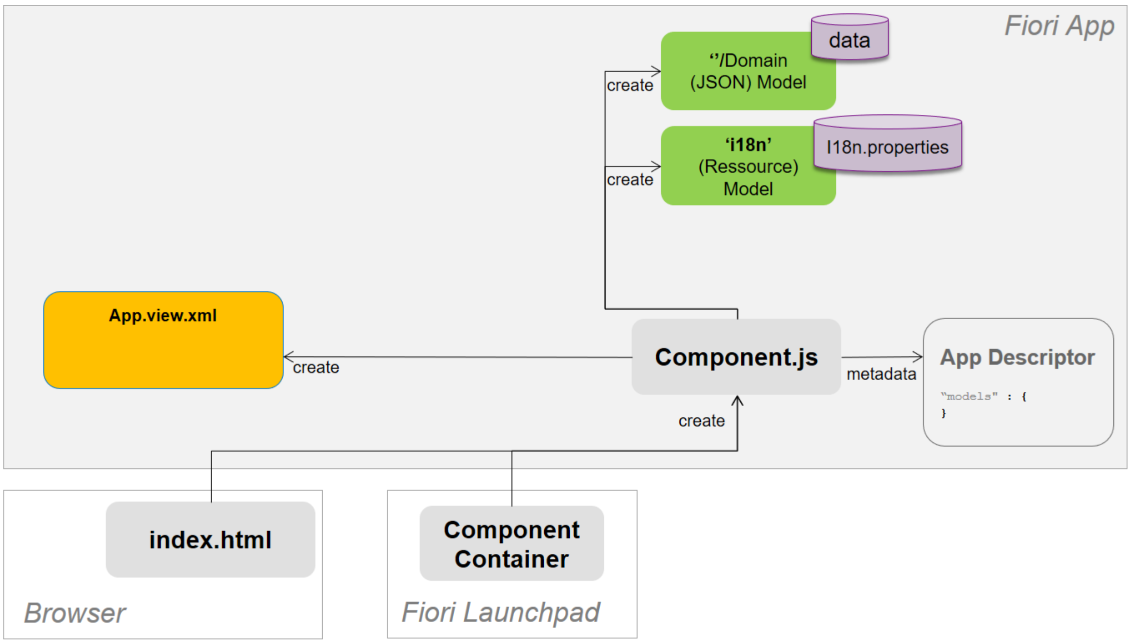


Figure 5.6: An illustration of how to load a Fiori application using Fiori Launchpad. How component controller initialization (Component.js) using the manifest.json (called App Descriptor in the figure) works and how it creates additional, essential, parts of an application such as root view (App.view.xml) and corresponding data models.

Bootstrapping

Bootstrapping is simply a technique to load and activate the libraries that are necessary for running the application. In the case of Fiori, the application downloads and initializes the SAPUI5 framework and the other libraries that are used and required in the framework. Once the download and initialization is complete, the Component.js file is loaded.

The entire bootstrapping process takes place in index.html, which then creates the necessary global variables that can be used generally across all application files.

Component.js

The Component.js file is the component controller and provides the runtime metadata and the component methods visible across all other view controllers. A component controller is defined with the asynchronous module definition (AMD) syntax. To create an SAPUI5 component, it is required to extend the UIComponent base class provided by SAPUI5 framework, that contains methods for initialization of runtime metadata and also returns the UI as a tree of SAPUI5 controls. Those runtime metadata are written in manifest.json file and must be defined as a property of component controller.

As mentioned, manifest.json contains properties that are used during runtime. Using this approach means developers need to write less application code, and can already access the information before the application is fully instantiated. Some attributes in the application descriptor (manifest.json) are just for information purposes, such as the minimum SAPUI5 version (minUI5version), others help external components (for example the SAP Fiori launchpad) to integrate the application correctly, but most of the attributes are actually used to configure specific aspects of the application that are needed frequently.

The most important configuration settings are:

- Models. Examples of models are the configuration of the OData service (default model) and language files (i18n model). All models described in the manifest.json file are automatically instantiated when the application is started.
- Libraries and components that are used in the application and have to be loaded during initialization.
- The root view of an application.
- Routing configuration that defines the navigation between views.

View

The App.view.xml file defines the root view of the application in general. In most cases, it contains an App control or a SplitApp control as a root control. SAPUI5 supports multiple view types (XML, HTML, JavaScript, JSON). XML views are most recommended, as for these developer has to separate the controller logic from the view definition. View is then implemented as an xml file containing tags that define the SAPUI5 user interface controls. This rule is then compiled to UI objects by using web browser build in compiler.

These controls (App, SplitApp) allow to create a Page control as its own aggregation. The App control takes and displays only single one Page control while SplitApp allows to display master-detail, that means, two Page controls are created at once. The application views then contain the parent Page control, which then encapsulate the other UI controls that define the shape and functionality of user interface. Navigation across the individual application views is then ensured by the Router object. Based on the URL, it destroys and creates a new application view, so the navigation takes place at the level of Page controls that are added or removed from the root view during the runtime by Router object created and defined in manifest.json file.

Controller

Controllers are implemented as classes that inherit from the Controller class defined by the SAPUI5 framework. Inheritance from this class provides event loop functionality, which are an events triggered when a controller is instantiated, before the view is rendered, after the view is rendered, and when the controller is destroyed. There are also important methods that make view available. This makes it possible to manipulate the controls that are contained inside the view, dynamically create, change, and delete data models. An important feature is also the ability to access the component controller instance (Component.js) and manipulate the variables defined in manifests.json.

Each controller is created as a class that contains the properties and methods for executing application logic, handling view events, and manipulating data models. This class is then defined using the asynchronous module definition (AMD) template. In this template, array of dependencies (paths to other files) whose functionality is utilized in the controller is first defined. Thanks to this dependencies, application logic can easily be divided into more classes where classes used as dependencies no longer have to inherit the properties from the SAPUI5 framework Controller class. In the body of the template is a complete definition of the view controller. The entire template is then initialized as a single class containing the properties of all dependencies by calling globally visible sap.ui.define method.

Model

A model in the SAPUI5 Model View Controller concept holds the data and provides methods to retrieve the data from the database and to set and update data. There are two types of models, server side and client side models. Server side model is called OData model. It contains the methods by which the communication between the backend OData service and the Fiori application is ensured. Each Fiori application contains an instance of at least one OData model. Using the implemented read, create, update methods, communication takes place via HTTP protocol and GET, POST, UPDATE and / or MERGE requests are generated with corresponding parameters.

The second type of model, the client side one, provides resources to pass data between the view and the controller. The most common type of this model is JSONModel which retains the data stored in JSON format, which is very similar format that javascript data structures are written in (objects, arrays). There is two-way client model, which allows to change the value of the model's properties both from the view and the controller plus there is one-way type, where the data inside the model can only be changed by the controller. An instance of such a model always comes from the application controller and must be explicitly binded to the appropriate view.

Another way to make data available for view or view controls is to use direct binding. In this way, the data entity, which is available through OData protocol, is directly binded into the view and / or the corresponding UI control. Data is then available in view without using client side model, the disadvantage being that it is more difficult to access and edit data inside the controller.

5.4.2 Architecture of Approve Bank Payments application

This section will briefly describe how the architecture has been adapted for developed application. The reasons behind these adjustments and the benefits that it brings will be mentioned. The main effort was to keep implementation of the application transparent and logically separate implemented functionality to make it easier for maintenance. The effort was also to identify the parts that could be used in many places and then adapt accordingly the architecture.

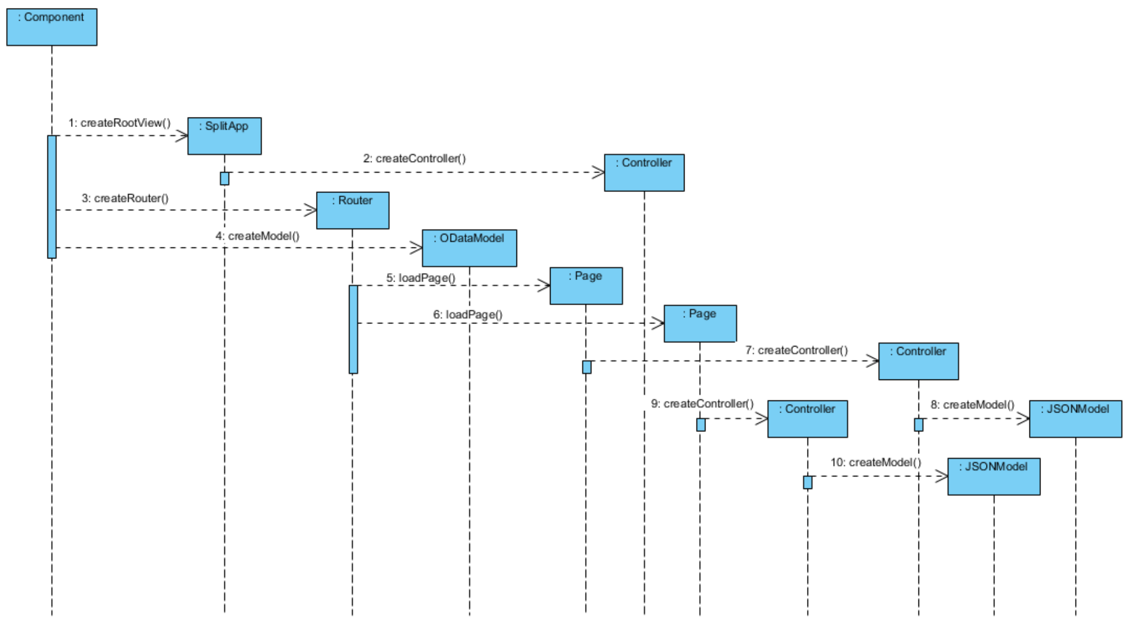


Figure 5.7: Sequence diagram illustrating how the most important parts of the application are created and initialized. Component controller creates root view, OData model and Router object. Using a Router object corresponding views that represents master and detail aggregation of SplitApp components is created based on the URL. Each view also creates the appropriate controller that initializes its data model.

View

As already mentioned, the SplitApp component was used as a root view. Other views to be used in the master or detail aggregation of SplitApp component are created using the Page component. The inner components of the view are the ones defining the layout of the page and the positioning of the inner components that visually represent the user data.

The components that represent this data are defined by fragments. This technique allows us to logically separate those parts of the application that are meaningfully similar from others. The fragment is, like the view, a XML file that uses tags to define the elements and their attributes that will be part of the UI. There can be added already defined fragment to a particular view by using Fragment component with the path attribute to the defined fragment to become part of the view.

Additionally, it is also possible to add an existing fragments to view dynamically. This is done by a controller. There is passed fragment path to the controller and then it creates an object from it, this object is then added as a child object to the view and / or another component inside the view. This technique is used when working with dialogs that are dynamically created and destroyed and can be used across multiple views.

Controller

The controller is primarily designed to handle user interaction with the UI, process and move data to the view and also execute application logic. For Approve Bank Payments, the controller inherits and extends the SAPUI5 Controller class. However, if all the application logic is contained in one controller, it may seem very large and confusing. Adding new functionality and maintaining the current would become very demanding.

For this reason, application logic has been split into several files. Within these files there are designed classes that provide a portion of application logic. The classes are made up by inheritance and extension of SAPUI5 Object class. The application logic was divided as follows, each view, that was created, contains the appropriate controller, with exactly the same name, that add application logic for a view. Very similarly are build other classes, every fragment that was created has a class that contains the methods with application logic of a fragment functionality. In this way, it is also possible to create separated classes that provide independent functionality which can then be used in any controller.

As a result, the controller includes implementation of methods that handle user events and communicate with the backend OData service. Implementation of methods providing application logic is moved to more classes. These classes are then added to the controller as its dependency.

Model

For Approve Bank Payments was created one OData model, defined in the manifest.json file, which makes it part of the component controller class and is therefore accessible to all application controllers that take control each view.

The remaining models are created by using the JSONModel class of the SAPUI5 framework. These models store the current UI state, its configuration, or the calculated values that can not be obtained directly from the entity set available through the OData protocol. There is stored information about which buttons should be visible, which list element is currently selected, the status of the application that indicates whether the user data is loaded, etc. Each view has created the corresponding data model (see table 5.1), because each screen contains different settings which need to be preserved and modified. To be able to customize the model for a specific view, it was created like a new class that inherits from the JSONModel class and contains all its methods and properties. All the methods to work with the settings of a specific view have been implemented inside newly created class.

JSONModel	**View file**	**Other files**
BatchViewModel	Batch.view.xml	Batch.controller.js BatchDraftcontroller.js
MasterViewModel	List.view.xml	Master.controller.js
PaymentViewModel	Payment.view.xml	Payment.controller.js
Models	-	Component.js

Table 5.1: Table shows existing JSON models and corresponding view that is model binded to. Last column shows another javascript files that manipulates JSON model.

This all means that user data is not moved into the view from its controller using the instance of model. Controls displaying data hold a fixed binding path to the entity set exposed by the OData protocol (see table 5.2). Since it is in many cases necessary for a data format to be different from what is provided by entity set, it was necessary to implement formatters. Formatter is a class implementing methods that format these data before they are rendered using UI controls. An example of a formatter can be a date formatter that is displayed differently in different countries.

Entity Set	SAPUI5 Control	File
C_AbpInstructionKey	SmartTable	EditInstructionKeyDialog.fragment.xml
C_AbpPayment	SmartTable	Payments.fragment.xml
C_ApbInvoice	SmartTable	Payment.view.xml
C_AbpPaymentBatch	List	List.view.xml

Table 5.2: Table showing existing entity sets provided by OData service trough OData protocol and UI components that shows provided data. Last column mention view files that contain corresponding SAPUI5 control.

5.5 Implementation

In this section, called Implementation, main parts of the application will be described in more detail. There will be described the necessary steps to make the application accessible to end users in the Fiori Launchpad environment (section 3.5.1).

There will be space for the most important UI elements that make up the individual screens of the application and the key functionality that was implemented in the controller will be mentioned as well.

Another subsection will also be a description of the implementation of CDS views that access the data stored in the SAP HANA database for backend ODATA service. The structure of CDS views and their hierarchical structure will be introduced.

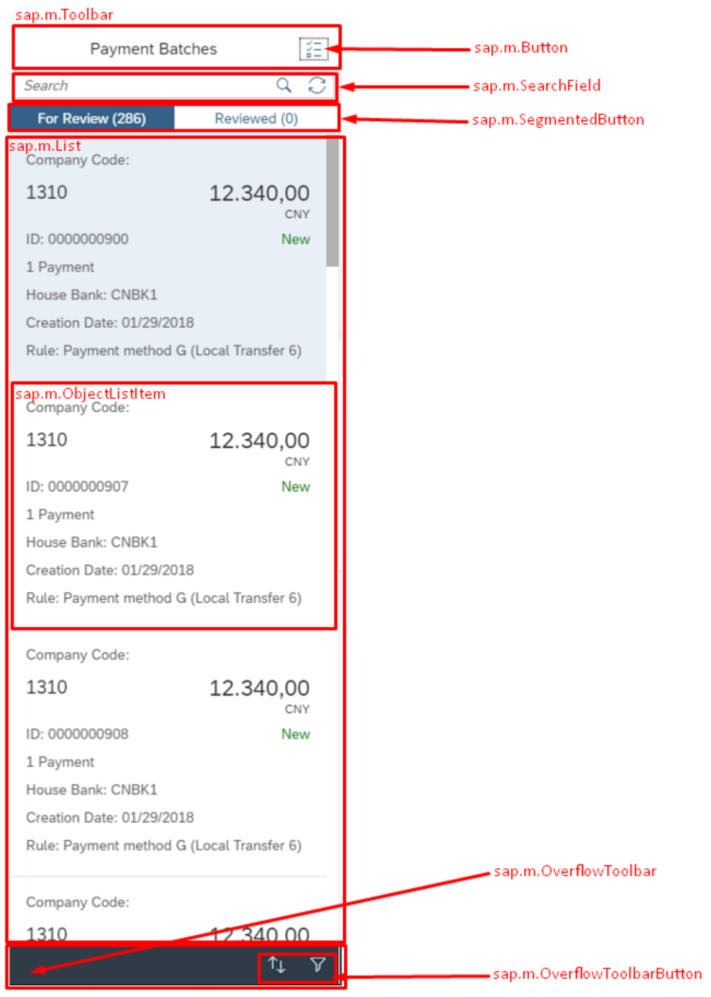
Last but not least, the Router object, which is used to navigate between individual application screens, will be described and important libraries that have been implemented and adapted to the application to be able to use the functionality they offer.

5.5.1 Accessing the application

In order to make application visible for the end-user, it is necessary to perform initial user settings. Developed Fiori applications could be visible from the Fiori Launchpad. From this user environment, the application can then be accessed by clicking the tile with its name. Users in the cloud environment, provided by SAP, have assigned user roles. Individual roles could be assigned to particular users using account with administrator rights via Maintain Roles application. However, the individual roles are built, among other things, from business catalogs. These business catalogs contain references to the technical catalog. In this technical catalog, there are finally all Fiori applications that are available and visible from SAP Gateway (see 3.2.3). Approve Bank Payments, available in technical catalog SAP_TC_FIN_FO_COMMON, is referenced into two business catalogs because it is designed for two different user roles and they have different data permissions. For the first approver (accounts payable manager role) is prepared catalog called SAP_SFIN_BC_AP_PAY_APV and for the second one (cash manager role) catalog called SAP_SFIN_BC_BANK_APPROVE. The first is then written to a role named AP_MANAGER and the other is for CASH_MANAGER role. The administrator can then assign these roles to the users responsible for approving bank payments.

5.5.2 List of payment batches

This subchapter describes the customization of the screen for displaying list of payment batches (figure 5.8). The screen header is done by Search component, which is a text



Company Code:
1310 12.340,00 CNY
ID: 0000000900 New
1 Payment
House Bank: CNBK1
Creation Date: 01/29/2018
Rule: Payment method G (Local Transfer 6)

Company Code:
1310 12.340,00 CNY
ID: 0000000907 New
1 Payment
House Bank: CNBK1
Creation Date: 01/29/2018
Rule: Payment method G (Local Transfer 6)

Company Code:
1310 12.340,00 CNY
ID: 0000000908 New
1 Payment
House Bank: CNBK1
Creation Date: 01/29/2018
Rule: Payment method G (Local Transfer 6)

Company Code:
1310 12.340,00

Figure 5.8: Overview of main controls that are visible on master area of application user interface.

input that can be used to search for a specific payment batch. Right below is the SegmentedButton component, which indicates whether the batches to be approved or already approved are visible. The main page control is a List. It displays payment batch items. There is used ObjectListItem layout control that determines the layout and appearance of individual list item information. ThefFooter area, created by OverflowToolbar, contains OverflowToolbatButtons that can be used to sort or filter over the List.

The controller implements the functionality by which the entity set C_AbpPaymentBatch is binded to the List control. The search operation then dynamically adds and removes the parameters of this binding between the control and the entity set. In the same way, it is implemented when switching between approved and rejected payment batches. When list item is clicked, the application URL is dynamically changed. At this URL address, the value of the selected payment batch identifier is changed. This change is registered with a Router object which, based on the URL, invokes the action for the data displayed on the Payment Batch detail screen. Sorting and filtering actions are done by predefined dialogs with sorting or filtering options over all the data that is visible on the payment batch.

These actions are implemented by controller methods without modifying the relationship between the list control and the entity set. This results in faster list response.

5.5.3 Payment batch detail

The top of the Payment Batch Detail screen (figure 5.9) consists of the ObjectHeader component. It makes it easy to display payment documents. It contains title and total properties that can highlight the name of the payment document and its final amount. In the body of this component, ObjectAttribute is used to create a responsive fragment of detailed payment batch information. Information Due Date, House Bank, and Account are made up of a Link component to capture click events on these values. At the bottom of the ObjectHeader, IconTabHeader, which serves as a menu that determines the content of the bottom of the screen, is inserted.

This section may display SmartTable showing individual payments. An important role here is played by OwerflowToolbar containing buttons for handling selected payments and table settings. This section also includes the TimeLine component or the List component that contains the attachment names and their links using the CustomListItem view template.

The bottom of the screen is finally created by the OverflowToolbar component, which contains buttons that manipulates the entire payment batches.

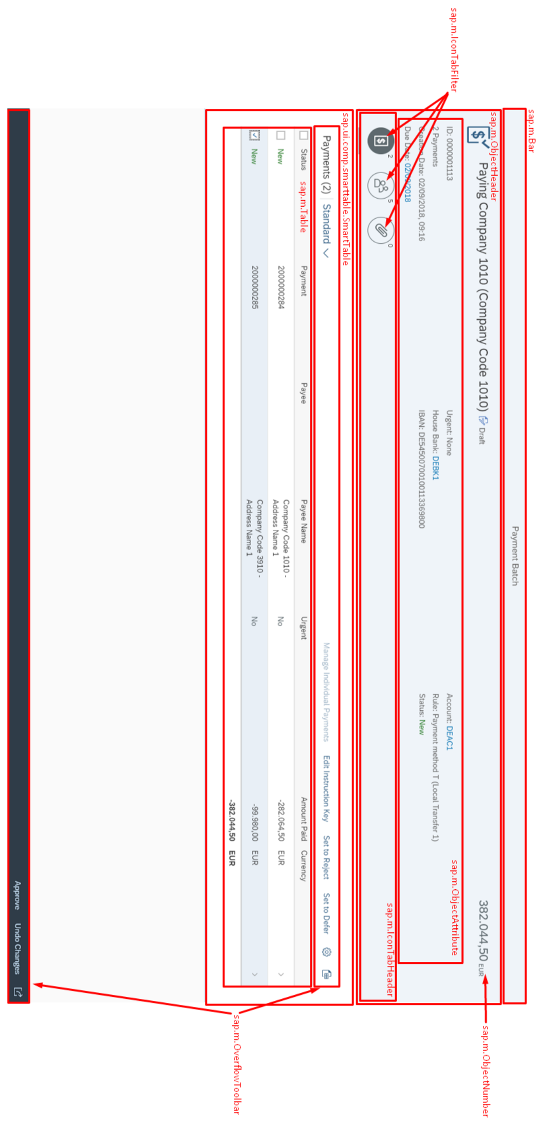
For the Due Date, House Bank, and Account attributes is dialog box dynamically created when a link of an attribute is clicked. For the Due Date attribute, the entity set C_AbpInstructionKey is red to be able to select appropriate value. Other dialogs show detailed information describing the attribute. The bottom of the screen changes and loads data depending on the option selected from IconTabBar. The payment table is displayed by default, and the data for other tabs is loaded once the tab is selected and displayed. All three fragments, that could be displayed by the user, are stored in memory after reading, including the data they display so they do not need to be reloaded while switching between fragments.

A JSONModel is created for the SmartTable component, which maintains the state of the selected lines. Individual selected payments could be approved or rejected, or a combination of both. By JSONModel inner state then enabled and disallowed the use of individual buttons so that data inconsistency can not be caused by a mishandling of the application. On the fragment showing attachments are controls that allow users to edit only attachments they are created and attached to the payment batch by them.

5.5.4 Payment detail

The Payment Detail screen (figure 5.10) is the same as the one above, the Payment Batch Detail, made up of the ObjectHeader to display a payment document. The difference is that instead of the IconTabHeader control, there is a Bar component containing SAPUI5 Select. There is then option to approve or reject the specific payment displayed using this Select. The bottom part is then a SmartTable component in which the invoices, related to the payment, are visible. The Journal Entry column values in the table are created by a Link component that allows you to navigate to Manage Journal Entries application to view the details of this document. The OverflowToolbar above the table allows to customize or export table to excel file.

On this screen, user can manipulate the displayed payment with the Select control. The value selection is captured and there is a logic that changes PaymentAction attribute



Payment Batch

Paying Company 1010 (Company Code 1010) Draft — 382.044,50 EUR

ID: 0000001113
2 Payments
Creation Date: 02/09/2018, 09:16
Due Date: 02/09/2018

Urgent: None
House Bank: DEBK1
IBAN: DE54500700100113369800

Account: DEAC1
Rule: Payment method T (Local Transfer 1)
Status: New

Payments (2) Standard

Status	Payment	Payee	Payee Name	Urgent	Amount Paid	Currency
New	2000000284		Company Code 1010 - Address Name 1	No	-282.064,50	EUR
New	2000000285		Company Code 3910 - Address Name 1	No	-99.980,00	EUR
					-382.044,50	EUR

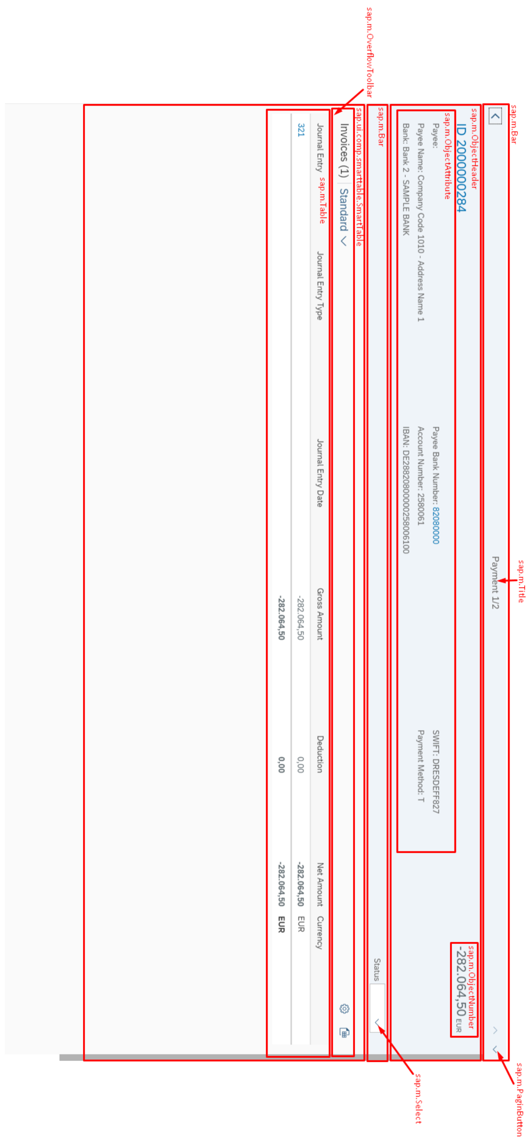
Figure 5.9: An image showing the user interface of the detail area composed of the mentioned controls.Image shows brief overview of controls visible for an end user. Including Payments.fragment.xml file with a table of payments is displayed.

of binded C_AbpPayment entity set to the SmartTable. Navigating to Manage Journal Entries to reach document details is ensured by changing the URL and adding identifiers that uniquely define the selected document (invoice in this case).

5.5.5 Router object

SAPUI5 comes with a powerful routing API that helps to control the state of implemented application. In classical web applications, the server determines which resource is requested based on the URL pattern of the request and serves it accordingly.

In single-page applications, only one page is initially requested from the server and additional resources are dynamically loaded using client-side logic. The user only navigates



Payment 1/2

ID 2000000284

-282.064,50 EUR

Payee:
Payee Name: Company Code 1010 - Address Name 1
Bank: Bank 2 - SAMPLE BANK

Payee Bank Number: 82080000
Account Number: 2580061
IBAN: DE28820800000025800610

SWIFT: DRESDEFF827
Payment Method: T

Invoices (1) | Standard

Journal Entry	Journal Entry Type	Journal Entry Date	Gross Amount	Deduction	Net Amount	Currency
321			-282.064,50	0,00	-282.064,50	EUR
			-282.064,50	0,00	-282.064,50	EUR

Figure 5.10: Image showing some really basic controls visible on the view of payment.

within this page. The navigation is persisted in the hash instead of the server path or URL parameters.

For example, a classical Web application might display the employee page when URL http://<host>/<app>/employee.html?id=3 or http://<host>/<app>/employee/3 is called. A single-page application instead would do the same thing by using a hash-based URL like http://<host>/<app>/#/employee/3.

The information in the hash, namely everything that is following the # character, is interpreted by SAPUI5 router.

Single-page applications based on SAPUI5 can use a so-called router object to dispatch hash-based URLs to one or more views of the application. Therefore, the router needs to know how to address and show the views. In SAPUI5, there is simply added a routing section into manifest.json file to configure the router. There are three properties that are used to configure the routing of application:

- Config: this section contains the global router configuration and default values that apply for all routes and targets. To load and display views automatically, we also specify the controlId of the control that is used to display the pages and the viewPath to tell router where all views are located.
- Routes: each route defines a name, a pattern, and one or more targets to navigate to when the route has been hit. The pattern is basically the hash part of the URL that matches the route. The sequence of the routes is important because only the first matched route is used by the router. The target property references one or more targets from the section below that will be displayed when the route has been matched.
- Target: a target defines the view that is displayed. It is associated with one or more routes or it can be displayed manually from within the application. Whenever a target is displayed, the corresponding view is loaded and added to the aggregation configured with the controlAggregation option of the control.

5.5.6 Draft 2.0

Draft is a technology that makes it easy to implement exclusive access to the data entities. This exclusive access is limited by a time limit, usually 20 minutes, and the created draft is then freed. A Draft creates a table containing deep copies of records that have been edited for a particular entity set (basic usage on figure 5.11). It allows the user to keep unsaved changes, to return to work later, to prevent the loss of unsaved changes when the application crashes, and also to indicate to other users who have a locked source and edit it at a given time.

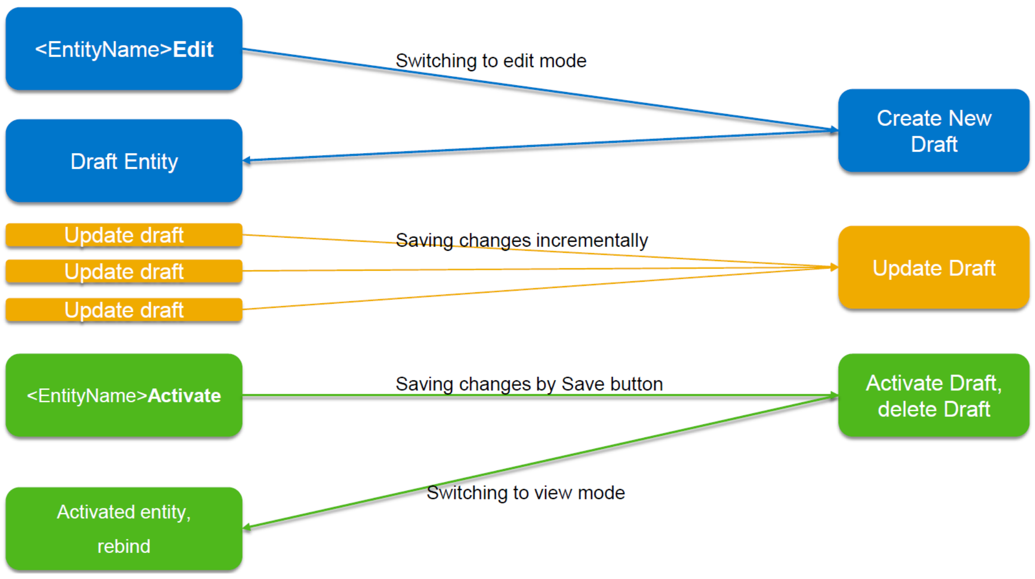


Figure 5.11: Image showing typical scenario how draft is used when implemented in application.

Creating a draft table is initialized from the application frontend part by setting the isActiveEntity attribute to that set entity. To the application is then returned access to the created draft table described using the DraftUUID attributes, isActiveEntity (indicating whether the draft table is still active) and PreserveChanges (setting value true to avoid creating another draft table). The library then provides save or cancel functionality to be able to save changes in created draft or delete draft and discard all unsaved changes.

5.5.7 Message popover

Message Popover is used in application to provide a list of different types of messages like errors, warnings, success and information type. Provides a handy and effective way to navigate and show details for every message returned from OData service.

The MessagePopover control displays a list of messages which can be further drilled down to reveal more details. It is placed in the footer of a detail area and can be expanded when clicking on its icon because control inherits from the sap.m.Popover. The MessagePopover control allows application to provide a long-text description for a message, which includes markup and formatting of the content.

Array of messages, which is displayed using the MessagePopover control, is stored in the JSON model named message. The footer are of views from Batch.view.xml and List.view.xml contain a button made up of the sap.m.semantic.messagesIndicator control with the callback function handleMessagePopoverPress registered to the press event. Within the callback function, a delegation of the program execution is passed to the Messaging.js object. This object includes the implementation of message processing methods and the creation of a MessagePopover control to display received messages.

5.5.8 CDS views implementation

CDS views are a tool that allows to select data from database tables. Once these views are created, they are registered to OData service and allow them to read or edit database tables. Five CDS views were created for Approve Bank Payments application, namely: C_AbpInstructionKey, C_AbpInvoice, C_AbpPayee, C_AbpPayment (figure 5.12), and C_AbpPaymentBatch. Such view could be divided into several parts that are more or less similar every time developing one.

The first part that each CDS view must contain is the annotation header. It provides several basic annotations, which may, of course, be extended if necessary. One of these is @AbapCatalog.sqlViewName, which defines the unique name of CDS view. Then @AbapCatalog.compiler.compareFilter defines that results from view could be filtered according to the exposed columns. Using @AccessControl.authorizationCheck, it is defined that when accessing a view, user authentication will be checked to be able to even display data. @AccessControl.personalData.blocking defines the information that the RAL (see 5.5.8) configuration has to be used to access some of exposed data and then the @AccessControl.privilegedAssociations annotation specifies which data sources this configuration will be configured for. On the view we have to additionally enable the update and delete operations using the ObjectModel.updateEnabled and ObjectModel.deleteEnabled annotations.

The second part, which is common to all CDS views, is its own definition. It is constructed using the following structure:

```
DEFINE VIEW viewName AS SELECT FROM sourceView
```

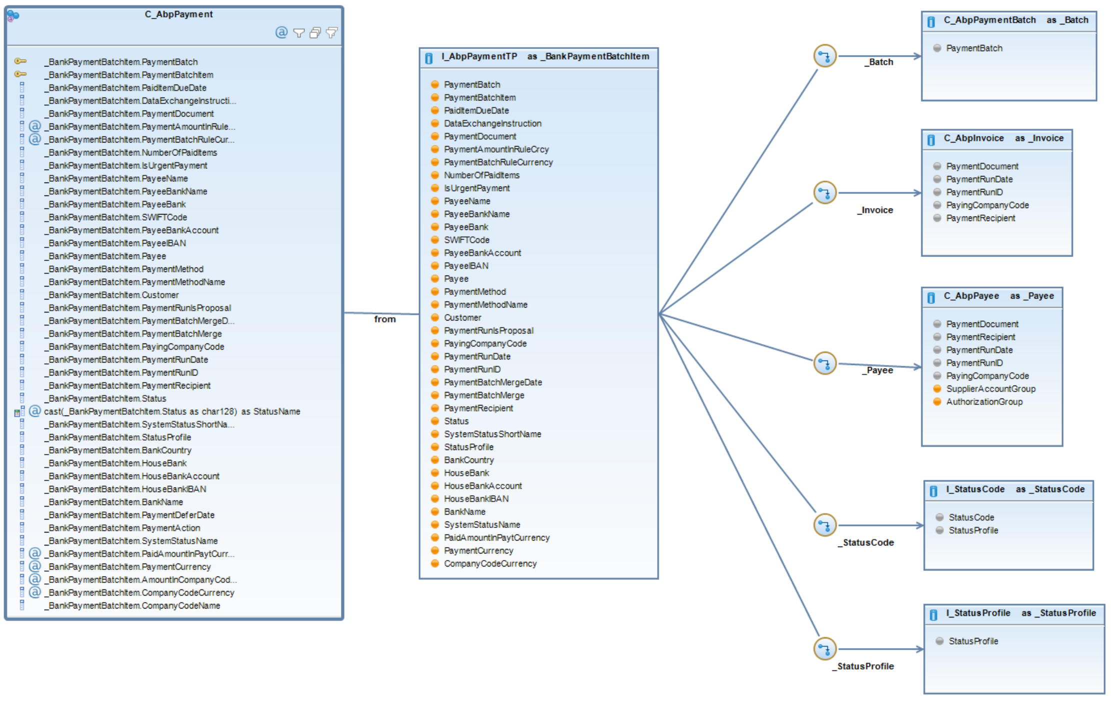

C_AbpPayment
_BankPaymentBatchItem.PaymentBatch
_BankPaymentBatchItem.PaymentBatchItem
_BankPaymentBatchItem.PaidItemDueDate
_BankPaymentBatchItem.DataExchangeInstructi...
_BankPaymentBatchItem.PaymentDocument
_BankPaymentBatchItem.PaymentAmountInRule...
_BankPaymentBatchItem.PaymentBatchRuleCur...
_BankPaymentBatchItem.NumberOfPaidItems
_BankPaymentBatchItem.IsUrgentPayment
_BankPaymentBatchItem.PayeeName
_BankPaymentBatchItem.PayeeBankName
_BankPaymentBatchItem.PayeeBank
_BankPaymentBatchItem.SWIFTCode
_BankPaymentBatchItem.PayeeBankAccount
_BankPaymentBatchItem.PayeeIBAN
_BankPaymentBatchItem.Payee
_BankPaymentBatchItem.PaymentMethod
_BankPaymentBatchItem.PaymentMethodName
_BankPaymentBatchItem.Customer
_BankPaymentBatchItem.PaymentRunIsProposal
_BankPaymentBatchItem.PaymentBatchMergeD...
_BankPaymentBatchItem.PaymentBatchMerge
_BankPaymentBatchItem.PayingCompanyCode
_BankPaymentBatchItem.PaymentRunDate
_BankPaymentBatchItem.PaymentRunID
_BankPaymentBatchItem.PaymentRecipient
_BankPaymentBatchItem.Status
cast(_BankPaymentBatchItem.Status as char128) as StatusName
_BankPaymentBatchItem.SystemStatusShortNa...
_BankPaymentBatchItem.StatusProfile
_BankPaymentBatchItem.BankCountry
_BankPaymentBatchItem.HouseBank
_BankPaymentBatchItem.HouseBankAccount
_BankPaymentBatchItem.HouseBankIBAN
_BankPaymentBatchItem.BankName
_BankPaymentBatchItem.PaymentDeferDate
_BankPaymentBatchItem.PaymentAction
_BankPaymentBatchItem.SystemStatusName
_BankPaymentBatchItem.PaidAmountInPaytCurr...
_BankPaymentBatchItem.PaymentCurrency
_BankPaymentBatchItem.AmountInCompanyCod...
_BankPaymentBatchItem.CompanyCodeCurrency
_BankPaymentBatchItem.CompanyCodeName
from
I_AbpPaymentTP as _BankPaymentBatchItem
PaymentBatch
PaymentBatchItem
PaidItemDueDate
DataExchangeInstruction
PaymentDocument
PaymentAmountInRuleCrcy
PaymentBatchRuleCurrency
NumberOfPaidItems
IsUrgentPayment
PayeeName
PayeeBankName
PayeeBank
SWIFTCode
PayeeBankAccount
PayeeIBAN
Payee
PaymentMethod
PaymentMethodName
Customer
PaymentRunIsProposal
PayingCompanyCode
PaymentRunDate
PaymentRunID
PaymentBatchMergeDate
PaymentBatchMerge
PaymentRecipient
Status
SystemStatusShortName
StatusProfile
BankCountry
HouseBank
HouseBankAccount
HouseBankIBAN
BankName
SystemStatusName
PaidAmountInPaytCurrency
PaymentCurrency
CompanyCodeCurrency
_Batch
C_AbpPaymentBatch as _Batch
PaymentBatch
_Invoice
C_AbpInvoice as _Invoice
PaymentDocument
PaymentRunDate
PaymentRunID
PayingCompanyCode
PaymentRecipient
_Payee
C_AbpPayee as _Payee
PaymentDocument
PaymentRecipient
PaymentRunDate
PaymentRunID
PayingCompanyCode
SupplierAccountGroup
AuthorizationGroup
_StatusCode
I_StatusCode as _StatusCode
StatusCode
StatusProfile
_StatusProfile
I_StatusProfile as _StatusProfile
StatusProfile

Figure 5.12: Example of database scheme for C__AbpPayment entity showing its attributes and associated internal views.

Where using the DEFINE VIEW command specifies the name of the created CDS view, this name will then be used through the OData protocol, which, for the frontend part of the application, describes the data model with the OData service is working. Finally, using the AS SELECT FROM command, we define the target data source providing data to our CDS view. However, the data that is available for Approve Bank Payments is not read directly from database tables, but from existing CDS views, these are called interface views and always start with the prefix I__. For all existing SAP HANA database tables, interface views, that make all their columns available, are created. This means that CDS views, created for a particular application, will never pick data directly from the database table itself but only from another existing interface view. The resulting views that are visible to applications are called consumption views and always start with the prefix C__.

Once the primary data source is defined using the select statement, the part, in which the associations are made with other source views, is followed. This association represents a JOIN operation and includes the cardinality of the relation as well as the conditions under which JOIN itself is performed. The syntax of such an association, which is part of previously defined view, then looks like this:

```
ASSOCIATION [min ... max] TO target AS associationName ON conditions.
```

The association attribute [min ... max] define the cardinality between the CDS view and the target data source, TO command is used to define the associated data source, consumption or interface view in most cases, the AS keyword defines the name of the connection under which the associated data can be accessed and ON keyword defines conditions and according to them is JOIN done (see table 5.3).

CDS view	**Source CDS view**	**Associations**
C_AbpInstructionKey	I_DataExchangeInstructionKeys	
C_ApbInvoice	I_PaymentProposalItem	I_Supplier
C_ApbPayee	I_PaymentProposlaHeader	I_AddressPhoneNumber I_AddressEmailAddress I_CountryText I_Supplier
C_AbpPayment	I_AbpPaymentTP	C_AbpPaymentBatch C_AbpInvoice C_AbpPayee C_AbpInstructionKey I_PaymentProposalHeaders
C_AbpPaymentBatch	I_AbpPaymentBatchTP	C_AbpPayment I_AbpHouseBank I_AbpBankAccount I_AbpRefNum I_HouseBank I_StatusCode I_StatusProfile I_Currency

Table 5.3: Table showing all existing consumption CDS views created for Approve Bank Payments application in the very first column. The second column mentions interface view from where are data selected and in the last column associated CDS views are mentioned.

The last part of the CDS view is its own body definition. It lists individual columns selected with the SELECT statement or through associations. Mentioned columns in the body are then visible to the OData service which can perform read, update, or delete operations above these columns. Additional annotations can be added to these exposed columns, the most common of annotations are @Consumption.hidden, which determines whether the column can be seen by the user and the @EndUserText.label annotation that specifies the text that will describe the column in the user interface.

Partial example of CDS view showing implementation of C_AbpInvoice from table 5.3:

```
@AbapCatalog.sqlViewName: 'CABPINVOICE'
@AbapCatalog.compiler.compareFilter: true
@AccessControl.authorizationCheck: #CHECK
@ClientHandling.algorithm: #SESSION_VARIABLE
@AccessControl.personalData.blocking:#REQUIRED
@Search.searchable: true
@VDM.viewType: #CONSUMPTION

define view C_AbpInvoice
as select from I_PaymentProposalItem as _PaymentProposalItem
          association [0..1] to I_Supplier as _Supplier
          on _Supplier.Supplier = _PaymentProposalItem.Supplier
{
     key _PaymentProposalItem.PaymentDocument,
     key _PaymentProposalItem.FiscalYear,
     key _PaymentProposalItem.AccountingDocumentItem,
     key _PaymentProposalItem.PaymentRunDate,
     key _PaymentProposalItem.PaymentRunID,
     key _PaymentProposalItem.PayingCompanyCode,
     key _PaymentProposalItem.AccountingDocument,
     @Consumption.hidden: true
     _PaymentProposalItem.PaymentRecipient,
     @Consumption.hidden: false
     @EndUserText.label: 'Net Amount'
     _PaymentProposalItem.NetAmountInTransacCurrency
}
```

5.5.9 Read Access Logging configuration

Read access logs represent business-relevant information that refers to a specific user. Therefore, it needs to be ensured that the creation of such information is protected against cross-site forgery. Since GET-requests do not provide XSRF-protection then any request, that is subject to read access logging, needs to be wrapped into a $batch request.

As soon as an active read access log configuration is created for a service then service metadata document changes. It shows the annotation USE_BATCH at the entity container indicating that the client is supposed to deliver $batch requests only.

Configuration takes place in a backend system using the SRALMANAGER transaction (understand as application in SAP backend system). This transaction simply allows to drag fields (columns) defined in CDS view into the RAL table to provide their registration to the RAL framework. Once configuration is activated, framework will create an access log on the system when reading previously registered fields.

5.6 Testing

There is a wide range of tests that can be used to verify the implementation of the application or to make it easy to maintain. It is possible to verify the correctness of the implemented methods, user interface behavior or meeting customer requirements. It all depends on what tests will be used. In this chapter we will mention unit tests that will be used for refactoring or minor change of functionality. Additionally, automated tests that run at regular intervals, and can capture errors, such as deploying a new framework version or adding new functionality. Using performance tests, you can measure the application response time when communicating with the backend, and eventually authorization tests confirm or disprove the logical execution of the application.

5.6.1 Unit and Opa5 tests

The goal of unit tests is to write an automated test for a certain program functionality. Then, we can them run unlimitedly times and validate code functionality after refactoring. Unit tests serve the programmer as an instant feedback to just written and committed code. Tests will be used to test smaller source code units. In development, it worked so that commit always contained implemented source code and relevant unit tests. Each time a code was written, then tests were immediately written as well. It is important that they are written immediately as long as the programmer has a logic function fresh in mind. The test should verify the behavior of the code both in standard situations and in extraordinary situations. E.g. what happens if the input method gets null, undefined or any other unexpected parameters. The purpose of the Opa5 (stands for One Page Acceptance) tests is then to check the inputs received by the individual user environment controls as well as outputs which will be returned after the input processing. In this way, it is possible to verify the correctness of the data transmitted to the UI controls by methods as well as the correctness of the output of UI controls which serves as the input parameter of the implemented method.

Tests for Approve Bank Payments are written using a framework called Gherkin[1]. Writing unit tests requires detailed knowledge of the platform on which the application is being developed, as well as a deep knowledge of the programming language and possibly other libraries. But most often, this knowledge may not be enough to understand the purpose of each unit test. Especially when working in a team, when the development and maintenance of the application is controlled by different groups of people, this problem is in place. The framework then allows tests to be written as simple and comprehensible sentences. Tests are created by using a .feature file, which is a template file that specifies how particular tests will be executed. They are written as consecutive sentences describing the test. There are individual scenarios in the template that form the test template. The script is a complex test of selected functionality, it consists of several consecutive sentences, a sentence should be described in the way like: click on a button, entering text into, etc., as well as a result such as a page redirection, or opening dialog. A corresponding template/sentences is/are implemented in the file, that, in this case, is written in JavaScript programming language. This file matches methods for individual sentences written in a file with the .featrure extension. The sentences are then replaced by the appropriate source code methods that implement unit or Opa5 tests.

Such tests are also very easy to read for a person who is not familiar with the technology in which the application is implemented. It is possible to find out which parts of the application are critical and what functionality tests cover. Which is advantageous, for example, for a tester who is not a developer of application, but will have an age of maintenance. For other developers it is appropriate for not having to read a lot of code for a long time. It can happen that the refactoring of the application is done by someone other than the one who actually wrote it, thanks to this verbal description of tests, it is possible to quickly identify the place for which the tests of the refactoring code are written.

Using unitTests.gherkin.qunit.html, unit tests could be run to control the functionality of selected application the most important core methods that should not be changed or overwritten during refactoring at all. In the integrationTests.gherkin.qunit.html file, Opa5 tests could be run, tests emulate the application itself, and using the defined scenarios it is possible to verify the functionality of the UI. It is also possible to test the inputs that the UI controls accept and the outputs they return.

5.6.2 Automation tests

Automation tests are one of the advanced techniques used to implement new software. It serves as a tool for both developers and testers. They are ran at regular intervals, either when committing a new changes or in a constant time interval, such as once a day, every two days or another. These tests primarily serve to verify the functionality of each element

[1]Gherkin test framework: https://github.com/cucumber/cucumber/wiki/Gherkin

of the user interface. There should be included key components that are often used and play an important role when using the application.

Individual tests can be understood as scenarios that are performed on a running application, and their feasibility is evaluated. The information that is very important is, of course, the results of the test, if all the tests have passed successfully or vice versa. If the test fails in any part of the action, it is important to identify a collision point in order to fix the error.

Robot framework[2] was designed to implement automated tests and its core programming language is Python. Its big advantage is, that it provides a wide range of predefined functions, these functions takes passed parameters most often in the form of a screen element identifier. These functions could be gradually composed and create test scenarios. If any function of the scenario is unsuccessful, execution will be interrupted and the new scenario will begin. The great advantage is, that these scripts can be fired on different systems and in different web browsers. Scenarios can also be parameterized and logically separated from each other, for example, determining which scenarios are testing data model and which are testing functionality of the user interface controls.

The framework automatically generates results.html and log.html files after the script ends. The results.html file generally contains information about how many scenarios have been successful and how many have ended unsuccessfully. It also provides statistics on the execution time of individual scenarios and the total run time. The log.html file provides more detailed view on the progress of individual scenarios. If a scenario has not been performed successfully, we can find an error statement describing the cause and screenshot of the application state prior to the occurrence of an error.

Tests are currently run every midnight and are tasked with verifying the OData service data model and the form of the data which OData service returns. The frontend part is closely related to the provided data model and any change could affect its functionality. Then there are tests that verify the functionality of the relationship between master and detail view.

5.6.3 Performance tests

Performance tests serve as a tool to check that the implemented application meets the limits set across all Fiori applications that are delivered to customers. Tests should check the application's weaknesses, these are the actions where the frontend part communicates with the backend of the application. These operations could transfer large amount of data and / or load dependent data. The location where dependent data are red can be recognized by the fact, that the frontend sends a query to retrieve the data that can just and only be used to obtain the target data that user requested. These operations are usually time-consuming. According to the prescribed rules, the application architecture should be designed so that a maximum of three dependent queries can be sent to obtain user data or perform other operations (tables 5.4 and 5.5 show results for Approve Bank Payment application).

First place that is measured, case one scenario, in terms of performance is application start. Rules say that only the most necessary data should be read after the application starts. By reading other, not user critical, data, the time when the application is fully usable for the end user is delayed and this reduces the resulting user experience. Measure

[2]Robot framework: http://robotframework.org/

counting begins by clicking the tile on the Fiori launchpad and takes until the libraries and the necessary data are loaded.

The second place, case two, that was designed as a risk and on which the performance measurement was done is the payment batch approval. In this case, one POST query indicating the payment batch approval is sent, then two GET queries are sent for re-rendering the modified batch list in the master view and the second, which receives the updated data displayed in the batch detail view.

The third case that is included for performance testing is action to display a payment detail, in which case two GET queries are sent to the OData service, which in the first case accepts the payment details that describe it, and secondly the invoices that are included in the payment. There could be up to several tens of invoices included in one payment. Tests were made on payments with twenty invoices.

The last test case is confirmation of all approved payment batches. Since such payment batches might be more, then so many POST operations will be sent as many payment batches were approved. All of these operations are sent at one time to the OData service in one $batch query. The server response is only sent when the last POST operation is processed.

Case	**Num. of queries**	**Num. of operations inside query**
1	3	10
2	3	12
3	2	3
4	2	6

Table 5.4: Table showing the complexity of queries when loading data. It outlines how many backend calls have to be made and how many operations these calls contain.

Case	**Average resp. time**	**Minimal resp. time**	**Maximal resp. time**
1	2,958 s	2,322 s	3,785 s
2	2,763 s	2,12 s	3,417 s
3	0,973 s	0,767 s	1,384 s
4	6,658 s	8,284 s	9,925 s

Table 5.5: Table showing performance test results for each case.

5.6.4 Acceptance tests

Acceptance tests primarily serve as a tool to verify the accuracy of the implemented functionality with respect to customer requirements for application. In addition to the functionality, the correct distribution of the visual elements and the logical structure of the application are verified. The goal is to ensure that the application is as comprehensible as possible, simple to use, provides clear information on key and necessary information, and last but not least, to ensure intuitiveness of available functionality for the end user.

Tests took place as a skype meeting with the selected customer. The customer was approached by product owner of an application. The customer has had to move around the financial sector for a while to ensure that the test results take into account his own experience and business knowledge.

These tests took place in two phases. In the first stage, there was in a discussion application prototype that contained only mock data and did not provide any data manipulation functionality. At this stage it was important whether the customer would be able to quickly and easily understand the information displayed on the screen. The conclusion of the first phase of the accessibility tests was, that some values in the list showing particular payment batches from the master view aggregation should be reassessed as more important than others. Which results in their highlighting within the item and thus ensuring that they are better visible and payment batches will be easier to identify and read. Specifically, this was a company code entry. Additionally, there was a rule entry that was missing in the previous proposal. On the payment batch detail screen, there was a request for the option to be able to select multiple payments at once for the possibility of approving or rejecting them.

In the second and also the final phase, the intuitive of the implemented functionality, that manipulates displayed data, was tested. Several key scenarios have been predetermined, such as showing a specific payment batches, approving a payment batch as a whole, defering specific payments, changing the due date value of the payment batch, and then displaying the detail screen of the selected payment. The conclusion of this test was that deferal and approval actions lack confirmation dialogs to check user decisions.

All the requirements were discussed with the product owner of the application. Suggestions for design changes were taken into account in the new design mockups developed by the application designer and subsequently implemented.

Chapter 6

Conclusion

In this thesis was explained and clarified that Approve Bank Payments will be used to validate and approve intra-company accounts payable that arise from vendor payments, internal payments, or travel expenses. Thesis also introduced the business logic behind payment processes and accounts payable.

SAP S/4HANA platform as it exists in SAP cloud environment was explained. It is represented by services that the platform provides to customers, a technological structure from SAP HANA database, through Core Data Services, which retrieve database data and access it through a service located in SAP Gateway that exposes data to client applications. Also there is mentioned OData protocol, which works like a description of data that is passed to client applications and is available in SAP Fiori Launchpad using tiles as an application entry points.

Subsequently, the most commonly used and the most suitable frameworks for creating the user interface were mentioned, namely those are ReactJS, Angular 2, Vue.js and SAPUI5. Frameworks are discussed from the work with data, structure and construction point of view.

At the end of the thesis, the development of the whole application is explained. It was created by the Design-Led Development Process methodology, which controls the development of all Fiori applications in SAP. The development is explained from the phase of mapping application's requirements. In this section was worked closely with the customer to cover all the necessary functionality that application should provide. Then has also been data provided by application analyzed to be able to design individual screens of application. Based on this, a detailed application design was created, including the SAPUI5 components taht could be found in the application.

At the implementation stage, the UI part, containing empty UI controls, was first processed. In addition, CDS views were implemented to retrieve data from the database that the backend OData service and the Fiori application itself are working with. In the final phase, the frontend part of the application was connected to the real data provided by the OData service. The main focus was on effective data loading, readability of the source code for future maintenance, and logical separation of implemented individual parts of the application.

Simultaneously with the implementation, OPA5 and Unit tests were written using the Gherkin framework to cover the implemented functionality. Upon completion, the heavy focus was put on testing the communication speed of the application when communicating with the OData service. There is still room for improvement and efficiency in the future.

Last but not least, acceptance tests were conducted directly with prospective users who tested the functionality and understanding of the application.

The application is currently deployed on internal testing cloud systems, and for users should be made available in August 2018 when a new cloud release is planned. Until then, the UI should no longer change. Some modifications could be expected in the form of OData service backend communication and OPA5 tests extension that validate the functionality of key SAPUI5 components that make up the user interface.

Bibliography

[1] *Approve Payments*. SAP Documentation. [Online; navštíveno 13.12.2017].
Retrieved from: https://help.sap.com/erp2005_ehp_06/helpdata/en/51/7ad0531d8b4208e10000000a174cb4/frameset.htm

[2] *Architecture Overview*. Angular. [Online; navštíveno 08.11.2017].
Retrieved from: https://angular.io/guide/architecture

[3] *Bank Communication Management*. SAP Documentation. [Online; navštíveno 13.12.2017].
Retrieved from: https://help.sap.com/erp2005_ehp_06/helpdata/en/60/7ad0531d8b4208e10000000a174cb4/frameset.htm

[4] *BANK COMMUNICATION MANAGEMENT – USER GUIDE*. SAP User Documentations. [Online; navštíveno 13.12.2017].
Retrieved from:
https://portal.wdf.sap.corp/irj/go/km/docs/corporate_portal/WS%20Finance%20%26%20Administration/Tools%20%26%20Resources/P2P_Tools_Resources/Accounts%20Payable/Guidelines%20%26%20Related%20materials/Bank%20Communication%20Management%20-%20User%20guide.pdf

[5] *Introduction*. Vue.js. [Online; navštíveno 05.11.2017].
Retrieved from: https://vuejs.org/v2/guide/#Getting-Started

[6] *Reactivity in Depth*. Vue.js. [Online; navštíveno 05.11.2017].
Retrieved from: https://vuejs.org/v2/guide/reactivity.html

[7] *SAP FI - Accounts Payable*. tutorials point. [Online; navštíveno 16.12.2017].
Retrieved from:
https://www.tutorialspoint.com/sap_fico/sap_fi_accounts_payable.htm

[8] *SAP Fiori - Introduction*. tutorialspoint. [Online; navštíveno 19.11.2017].
Retrieved from:
https://www.tutorialspoint.com/sap_fiori/sap_fiori_introduction.htm

[9] *What Fiori Means to SAP and Its Customers*. computer economics. [Online; navštíveno 19.11.2017].
Retrieved from: https://www.computereconomics.com/article.cfm?id=1952

[10] *What is Angular?* Angular. [Online; navštíveno 08.11.2017].
Retrieved from: https://angular.io/docs

[11] Aschmann, P.: *Despite little notoriety, SAP Gateway making a big impact.* TechTarget. [Online; navštíveno 2.1.2018].
Retrieved from: http://searchsap.techtarget.com/feature/Despite-little-notoriety-SAP-Gateway-making-a-big-impact

[12] Averkamp, H.: *Accounts Payable Process.* Accounting Coach. [Online; navštíveno 23.10.2017].
Retrieved from:
https://www.accountingcoach.com/accounts-payable/explanation/2

[13] Bisht, S.: *Core Data Services [CDS] in SAP S/4 HANA.* SAP. September 2016. [Online; navštíveno 02.12.2017].
Retrieved from:
https://blogs.sap.com/2016/09/26/core-data-services-cds-in-sap-s4-hana/

[14] Bragg, S.: *The invoice approval process.* Accounting Tools. May 2017. [Online; navštíveno 14.11.2017].
Retrieved from: https://www.accountingtools.com/articles/2017/5/5/the-invoice-approval-process

[15] Hunt, P.: *Why did we build React?* ReactJS. June 2013. [Online; navštíveno 03.11.2017].
Retrieved from: https://reactjs.org/blog/2013/06/05/why-react.html

[16] Instructor, M.: *Understanding Accounting: Accounts Payable Function.* MONEY INSTRUCTOR. [Online; navštíveno 28.10.2017].
Retrieved from:
http://content.moneyinstructor.com/1473/accountspayable.html

[17] Joseph, A.: *A simple overview on SAP Netweaver Gateway.* SAP Blogs. January 2013. [Online; navštíveno 08.12.2017].
Retrieved from: https://blogs.sap.com/2013/01/24/a-simple-overview-on-sap-netweaver-gateway/

[18] Králová, I. K.: *Závazky z pohledu účetnictví.* portal.pohoda.cz. September 2013. [Online; navštíveno 17.10.2017].
Retrieved from: https://portal.pohoda.cz/dane-ucetnictvi-mzdy/ucetnictvi/zavazky-z-pohledu-ucetnictvi/

[19] Kurian, G. G.: *How Virtual-DOM and diffing works in React.* Medium. January 2017. [Online; navštíveno 03.11.2017].
Retrieved from: https://medium.com/@gethylgeorge/how-virtual-dom-and-diffing-works-in-react-6fc805f9f84e

[20] Lohrey, J.: *Accounts Payable Department Functions.* Chron. [Online; navštíveno 09.11.2017].
Retrieved from: http://smallbusiness.chron.com/accounts-payable-department-functions-73537.html

[21] OASIS: *OData Version 4.0. Part 1: Protocol Plus Errata 03.* OASIS. June 2016. [Online; navštíveno 05.12.2017].

Retrieved from: http://docs.oasis-open.org/odata/odata/v4.0/odata-v4.0-part1-protocol.html

[22] OASIS: *OData Version 4.0. Part 2: URL Conventions Plus Errata 03*. OASIS. June 2016. [Online; navštíveno 05.12.2017].
Retrieved from: http://docs.oasis-open.org/odata/odata/v4.0/odata-v4.0-part2-url-conventions.html

[23] Pandey, N.: *Create OData endpoint for MongoDB on MEAN stack*. CODE PROJECT. November 2016. [Online; navštíveno 07.12.2017].
Retrieved from: https://www.codeproject.com/Articles/1111490/Create-OData-endpoint-for-MongoDB-on-MEAN-stack

[24] saiprasadbagrecha: *SAP FI - Account Payable (AP)*. LinkedIn. January 2013. [Online; navštíveno 20.11.2017].
Retrieved from: https://www.slideshare.net/saiprasadbagrecha/sap-fi-account-payable-ap

[25] SAP: *ABAP CDS - Annotations*. SAP Help Portal. [Online; navštíveno 4.1.2018].
Retrieved from: https://help.sap.com/doc/abapdocu_750_index_htm/7.50/en-US/index.htm?file=abencds_annotations.htm

[26] SAP: *CDS Associations*. SAP Help Portal. [Online; navštíveno 4.1.2018].
Retrieved from: https://help.sap.com/viewer/b3d0daf2a98e49ada00bf31b7ca7a42e/2.0.02/en-US/6fcd6e5883f04de5b618a6d91141afb4.html

[27] SAP: *Design-Led Development Process*. Fiori Design Guidelines. [Online; navštíveno 20.12.2017].
Retrieved from: https://experience.sap.com/fiori-design-web/design-led-development-process-external/

[28] SAP: *SAP Web IDE*. SAP. [Online; navštíveno 18.12.2017].
Retrieved from: https://www.sap.com/developer/topics/sap-webide.html

[29] Sharma, T.: *ABAP Core Data Services – Part 1(ABAP CDS Entities)*. September 2017.

[30] Uhlíř, I. M.: APPLICATION FOR RISK MANAGEMENT REPORTING IN SAP. 2016.

[31] Wheeler, K.: *Learning React.js: Getting Started and Concepts*. scotch.io. October 2014. [Online; navštíveno 03.11.2017].
Retrieved from: https://scotch.io/tutorials/learning-react-getting-started-and-concepts

[32] Wong, K. W.: *Other then ABAP, can CDS consumed by others reporting tools?* SAP Archive. April 2016. [Online; navštíveno 21.11.2017].
Retrieved from: https://archive.sap.com/discussions/thread/3834390

[33] Xourse: *ReactJS App Architecture*. Medium. January 2017. [Online; navštíveno 03.11.2017].

Retrieved from:
https://medium.com/@Xourse/reactjs-app-architecture-7a33d7ae9834

Appendix A
Content of attached CD

- folder application/ - application’s frontend source code,
- folder views/ - implementation of CDS views,
- folder roboTests/ - source code of automation tests written in Robot framework.