JOHANNES KEPLER
UNIVERSITY LINZ

Author
Günther Haffner, BSc MBA

Submission
INSTITUTE OF NETWORKS AND SECURITY

Thesis Supervisor
Assoc.Prof. Mag. Dipl.-Ing. Dr. Michael Sonntag

August 2016

DEVELOPMENT AND INTRODUCTION OF A WEB BASED SOFTWARE FOR THE ADMINISTRATION OF INVENTIONS AT THE UNIVERSITY

Master's Thesis
to confer the academic degree of
Diplom-Ingenieur
in the Master's Program
Computer Science

Abstract

The aim of this thesis is the development and introduction of a web based software for the administration of inventions at the university. The current process of administration is done within Microsoft Excel, which leads to problems like missing deadline management and unstructured data.

Beginning with the requirements to manage inventions from the disclosure to the potential revenue of selling rights on patents we looked at various solution paths to create a software to replace the Microsoft Excel sheets used. After deciding about the architecture and technologies to use we implemented the solution as multi-tier-application. All data is stored in a DBMS. The application server was implemented using Java. The server offers a REST-API and decouples the client completely from the server. The client was developed as Angular application and can be used within a modern web browser.

The end users were involved through the whole phase of implementation. The agile development process enabled the end users and development team to closely work together and adopt the product to the needs of the end users. The challenge to allow the login with the university Single Sign On could be solved.

The actual use of the software and automated testing raised the confidence in the created solution. In comparison to commercial products the delivered software can be used without third party license and maintenance costs. The solution can be extended to deliver a solution for other universities too.

Zusammenfassung

Der Zweck dieser Masterarbeit ist die Entwicklung und Einführung einer web-basierten Software zur Verwaltung von Erfindungen an der Universität. Die Verwaltung wird derzeit mit Hilfe von Microsoft Excel durchgeführt, was zu Problemen wie fehlende Erinnerungsmöglichkeit oder unstrukturierte Daten führt.

Beginnend mit den Anforderungen zur Verwaltung von der Erfindungsmeldung bis zu den möglichen Erlösen aus dem Verkauf von Rechten an Patenten haben wir uns verschiedene Lösungsmöglichkeiten für den Ersatz der Microsoft Excel Arbeitsblätter angesehen. Nach der Architektur-Entscheidung und der Auswahl der Technologien für die Lösung haben wir eine Multi-Tier-Applikation entwickelt. Die gesamten Daten werden in einem DBMS gespeichert. Der Applikationsserver wurde mit Java implementiert. Der Server stellt eine REST-API zur Verfügung und entkoppelt dadurch den Client komplett vom Server. Der Client wurde als Angular Applikation entwickelt und kann in einem modernen Web-Browser verwendet werden.

Die Benutzer waren während der gesamten Zeit in die Entwicklung eingebunden. Der agile Entwicklungsprozess ermöglichte eine enge Zusammenarbeit zwischen den Benutzern und dem Entwicklungsteam und erlaubte die Anpassung des Produkts an die Anforderungen der Benutzer. Die Herausforderung, das Login in die Applikation mit dem Universitäts-Account (Single Sign On) zu ermöglichen, wurde erfolgreich bewältigt.

Die Benutzung der Software durch die Anwender sowie automatisiertes Testen erhöhten das Vertrauen in die gelieferte Lösung. Im Vergleich mit kommerziellen Produkten kann die gelieferte Software ohne zusätzliche Lizenz- und Wartungsgebühren eingesetzt werden. Die Lösung kann erweitert werden und so auch für andere Universitäten eingesetzt werden.

Table of Contents

List of Abbreviations

ACID. *Atomicity, Consistency, Isolation, Duration*
AJAX. *Asynchronous JavaScript and XML*
ASL. *Apache Software License*
aws. *Austria Wirtschaftsservice Gesellschaft mbH*
COTS. *Commercial off-the-shelf*
CSS. *Cascading Style Sheets*
DBMS. *Database Management System*
EPA. *Europäisches Patentamt*
EPO. *European Patent Organisation*
FUN. *Abteilung Forschungsunterstützung*
HTML. *HyperText Markup Language*
HTTP. *HyperText Transfer Protocal*
IPR. *Intellectual Property Rights*
JAXB. *Java Architecture for XML Binding*
JAX-RS. *Java API for RESTful Web Services*
JDBC. *Java Database Connectivity*
JKU. *Johannes Kepler University*
jOOQ. *Java Object Oriented Querying*
JSF. *Java Server Faces*
JSON. *JavaScript Object Notation*
JSP. *Java Server Pages*
MVC. *Model-View-Controller*
MVVM. *Model-view-viewmodel*
OEP. *oose Engineering Process*
ORM. *object relational mapper*
OS. *Operating System*
PC. *Personal Computer*
PCT. *Patent Cooperation Treaty*
REST. *representational state transfer*
SPA. *Single Page Application*
SQL. *Structured Query Language*
TLS. *Transport Layer Security*
URI. *Uniform Resource Identifier*
URL. *Uniform Resource Locator*
XML. *Extensible Markup Language*

List of Figures

List of Tables

1 Introduction

"Next came the patent laws. These began in England in 1624, and in this country with the adoption of our Constitution. Before then any man [might] instantly use what another man had invented, so that the inventor had no special advantage from his own invention. The patent system changed this, secured to the inventor for a limited time exclusive use of his inventions, and thereby added the fuel of interest to the fire of genius in the discovery and production of new and useful things." Quote from Abraham Lincoln [1]

1.1 Motivation

The scientific staff of the Johannes Kepler University Linz (JKU) does research in a variety of different areas. Parts of this research are leading to inventions. Those inventions could potentially become patents. The university law reserves for the JKU the right to take up the invention within a predefined period of time. However, it can be difficult to decide upon this, so sometimes external evaluators are necessary. Finally, after the JKU takes the invention into its own portfolio, all further steps – the final goal is getting a patent – have to be done somehow. In today's daily business these arrangements are typically aided by a computer and appropriate software.

Currently the JKU manages this with the help of a Microsoft Excel file, one invention per row, including also all information regarding the invention. Important information like upcoming dates with tasks are colored red, abbreviations are used for different elements to keep the single cells in Microsoft Excel shorter. The information is structured with columns, however, several columns contain more than one information at the same time.

The administrative staff has to create reminders in a calendar software, because Microsoft Excel provides no such feature. There is no integration between an invention row in Microsoft Excel and a reminder in some other application. This can lead to differences between those two systems, because a human is still responsible to maintain the synchronicity of the two. Another major issue is the missing multi-user-capability of the Microsoft Excel file. Without this only one staff-member at a time can use the file.

1.2 Mission

The aim of this master thesis is to create a software for managing inventions and patents *especially for the JKU*. This new software should supersede the current Microsoft Excel sheets. The top requirements are the following:

- The JKU-internal workflow from the invention to a potentially granted patent, even to the sale of rights on the patent, should be handled completely within the software.
- The software should be able to be used by multiple users concurrently.
- The system should contain a notification service. This service should send e-mail-messages to predefined receivers, and those messages contain reminders about tasks

which have to be performed, such as checking whether the response from the evaluator arrived in time.

- There should be a reporting area, where different predefined reports can be accessed, e.g., the number of patent applications within a calendar year and the actual status of those applications.
- The system should be built with components and resources which are available for public and should not entail license costs.
- The software will be used on the computer systems of the JKU, which means it has to run on those systems, mostly on PC with Microsoft Windows, Linux or an Apple Computer with Mac OS. It is not required to be usable on mobile platforms like tablet computers.
- The information within the system (the data about inventions etc.) has to be stored within a regularly backup-process on a separate platform. In case of data loss it should still be possible to restore it.
- The authentication and authorization of users should take place within the centralized JKU infrastructure by using Shibboleth [2].

The required features about the necessary data-fields and reports are described by the documentation of the actual used Microsoft Excel file combined with wishes and enhancements from the user's perspective. Although these are not fully documented requirements they can be used as a starting point for a prototype, which could be further developed to the final solution.

Albeit the created software solution may be used by other universities or even companies, it is not in the scope of this thesis to produce a solution for them. However, the last chapter *Summary* will contain information about enhancements which may help to modify the solution to a broader customer base, especially other universities.

1.3 Overview

The thesis is structured within chapters, and this section shows the overview of the different chapters.

Chapter 2 (Theoretical Background) shows the current state of managing inventions and patents at the JKU and possibly available chargeable software solutions for managing those.

Chapter 3 (Solution-Overview) describes the possible solutions, the decisions and rationale which led to the final solution.

Chapter 4 (invadmin) provides an overview of the different programs. The architecture of the solution (called invadmin) is described and important implementation details are presented.

Chapter 5 (Evaluation) evaluates the solution with the usage of the software within the context of the administrative organization of the JKU. It also provides information about the usage on computers with low processing power and responsiveness with more data than typically expected.

Chapter 6 (Summary) gives an overview of the problems arose during the development of the solution. It also contains ideas of future changes and enhancements of the solution.

The *appendix* contains information about the database model, a detailed installation guide for the running system, a detailed installation guide for a developer system, a guide for the creation of a release build and a user documentation of the software system.

Although the thesis is written in English, there are some specific words, most of the time from the Austrian Law, which could not be exactly translated. In those cases the German words are additionally placed within the sentences to allow German speaking readers a better understanding.

We write in this thesis about inventors in singular for better understanding. However, inventions are not only ideas from a single person, most of the time there is a team included. In addition, a gender neutral writing was discarded in favor of a single male form. However, the intended meaning is always gender neutral.

2 Theoretical Background

The JKU employs more than 1800 scientists in Linz. During their research scientists may invent new products or procedures. At the beginning it is often unclear if it happens to be a new invention. However, the JKU has the right to take up ("Aufgriffsrecht") those inventions and try to get a patent for such inventions. [3]

The department of research support ("Forschungsunterstützung" or just FUN) is responsible for the management of the inventions from employees of the JKU. During the whole business process, which will be described below, the department is in charge of doing the necessary work and to meet all deadlines posed by different organizations, e.g. external evaluators, attorneys at law or patent offices.

In the following sections the business process from an invention to a patent will be shown. After the textual process description the management of the inventions and patents currently done at the JKU (named the "*JKU Excel solution*") will be presented, finishing with an overview of possible software solutions from the COTS-market.

A business process is a sequence of in time and scope contiguously activities to create a business relevant result. [4] The target result of our business process is to get a granted patent. [3] However, even if the JKU does not get a granted patent, they have to manage the process in a structured way.

2.1 Invention to patent – the business process

The process is a general one which means it is the same for an individual, a company or a university. However, the process at an Austrian university has some more steps involved, which are also described in the following. Nevertheless, those additional steps may also occur in companies or with inventions from individuals, the scope of the thesis is a software solution for the JKU.

2.1.1 Common process

At the very beginning the inventor has to have an idea (*the invention*). To be patentable this invention has to fulfill the following requirements: [5] [6]

- *"The invention has to fall within the scope of patentable subject matter as defined by national law. This is different from country to country."* [5]
- The invention has to be new (which means that no one has published this idea somewhere before)
- The technical state of the art is enhanced, and this is *"not obvious to anyone with a reasonably decent knowledge of the field".* [5]
- It can be industrially used or created. [6]

In addition to these restrictions there may be other rules in the patent law (which is national law, different by country) which prohibit granting a patent, or some do not even allow the research which led to the invention at all. [5]

This invention, if publicly available, can be used by anyone else without any concessions to the inventor, therefore it is necessary to protect the interests of the inventor. These could be of a different kind, starting with prohibiting anyone else to use his idea, up to granting rights to someone to use his idea, but in exchange for a reasonably amount of money. However, the inventor can also sell his idea to someone else.

During the last centuries, most or all countries around the world introduced a patent law. This national law is different between the countries, but one common intention is the protection of the inventors' invention and interests. The main subject is the patent, which *"is a right given by the government of a country to the inventor of a product or process in order to protect his rights in selling it in that region, without worrying about others simply copying it and setting up in competition without barrier, for a specified period of time. It provides the patent owner with the right to exclude others from making, using, or selling a claimed invention, i.e. an invention as described in the claims."* [7]

Before getting a patent, the inventor has to file a *patent application*. In this application, the invention has to be described in detail and claims have to be added and the application has to be submitted to a patent office. The patent office examines the application, and if everything is fine (the patent requirements are met) the office *grants a patent*.

With the definition of a patent in mind, the region (or country) is very important. If e.g. a patent was granted for Austria, there is no protection in Germany. This implies that everyone out of Austria can use the idea of the patent without being charged any fees by the Austrian inventor. To circumvent those effects it is necessary to either apply the same patent in several countries at the same time (with each country having different requirements), or use one of the special application processes (uniform filing, no matter what countries are targeted). Table 1 shows an overview of the available patent application processes.

Application	Description
National patent application	The patent application is valid for the country where the application is filed. Such a national patent office is e.g. in Austria the "Österreichisches Patentamt" in Vienna. When a national patent application is filed, the result is a national patent.
European patent application	The European Patent Organisation (EPO / EPA) allows the uniform filing of a patent application for more than 35 countries. [8] After filing and during

	the examination process the inventor can choose in which countries he wants to receive a patent.
International patent application	The Patent Cooperation Treaty (PCT) is a contract about international cooperation on patents between almost 150 countries. The treaty allows also the uniform filing of a patent application. On a later phase of the examination the inventor has to specify the countries he wants to receive a patent.

Table 1: List of available patent application processes [9]

All the application as well as a granted patent are only available if the applicant and patent holder pay different fees. After granting the patent, the patent is at most twenty years valid, in case the patent holder pays the regular fees.

2.1.2 JKU process

The university has published a process (Stage-gate-process) of how they proceed with inventions from employees. The process guides the involved parties and eventually helps the university to decide whether to take up the invention or not. Additionally, the university has also published an IPR strategy, where the tasks and stakeholders are defined. Within this IPR strategy the inventions and the protection of inventions with patents are one of the missions of the university. [10]

Although the common process cannot be very different for the JKU, it involves additional steps worth mentioning. The process also starts with the idea and the following invention. However, there has to be the notice in advance of the central authority on inventions at the JKU, the department of research support. This notification has to be done in written form with the filing of a predefined form.

In the Austrian patent law, inventions of employees are defined as inventions which may belong to the employer. [11] However, the employer has also the right to relinquish on an invention. It means the university has to decide either to take up the invention and try to get a patent, or leave it to the employee to obtain a patent on his own. The difference between the two possible decisions is, which party has to pay the fees and gets in return the rights and possible revenues. Both options can lead to a granted patent, in which the inventor would be the same, but the owner would differ.

To provide information and help deciding what the university should do external evaluators are contracted, e.g. the Austria Wirtschaftsservice Gesellschaft mbH (aws). They provide services to examine the invention and give advice to the university (payable service). [12]

If the university relinquishes the rights of the invention, no further administrative steps (except the formal written decision) have to be done by the university, the process itself is then finished.

In case the university avails its rights the application process starts. With the help of specialized attorneys the patent application is written and filed.

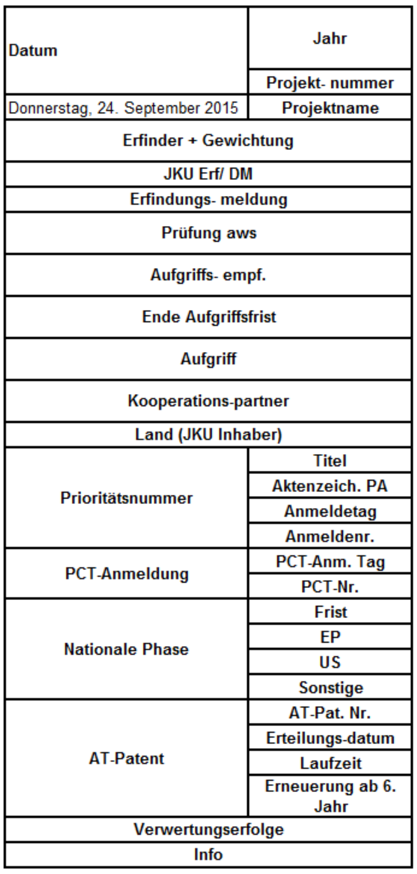
If the patent office(s) grants a patent, the university is the holder of the patent. The inventors may get remuneration, which is also specified in contracts with the employees and part of the Austrian patent law.

During the lifetime of the patent, the university receives invoices from different parties, e.g. the annual fee of the patent office, attorneys for writing requests to the patent office or a possible patent infringement case. On the other side the university also writes invoices, e.g. for legal use of the invention or a cooperation-fee for partners.

2.2 JKU Excel solution

The Excel solution is a file with two sheets: active and inactive projects. A project in this sheet is an invention and contains all accompanying data like patent applications and granted patents. The inactive state is used on inventions where the JKU relinquished the rights on the invention, or there are no pending patent applications and no valid granted patents anymore. All other inventions are active, which means there are still ongoing processes for those inventions.

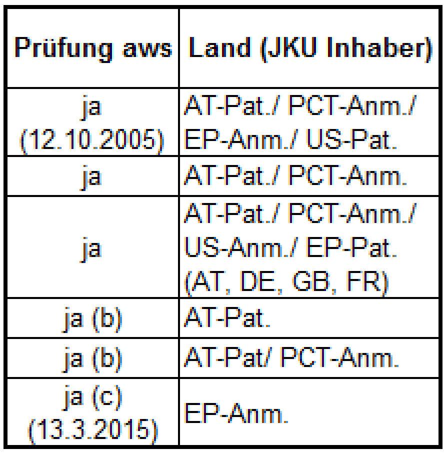
A sheet is organized in rows, columns and cells. The sheets start in the first rows with a header which names each column. The header of the Excel solution is shown in Figure 1 (transformed into lines). Afterwards each row contains information about a single invention. The columns structure the data to contain the same type of information within one column. However, the information within the cells is often unstructured data and cannot be processed automatically. [13]



Datum	Jahr
	Projekt- nummer
Donnerstag, 24. September 2015	Projektname
Erfinder + Gewichtung	
JKU Erf/ DM	
Erfindungs- meldung	
Prüfung aws	
Aufgriffs- empf.	
Ende Aufgriffsfrist	
Aufgriff	
Kooperations-partner	
Land (JKU Inhaber)	
Prioritätsnummer	Titel
	Aktenzeich. PA
	Anmeldetag
	Anmeldenr.
PCT-Anmeldung	PCT-Anm. Tag
	PCT-Nr.
Nationale Phase	Frist
	EP
	US
	Sonstige
AT-Patent	AT-Pat. Nr.
	Erteilungs-datum
	Laufzeit
	Erneuerung ab 6. Jahr
Verwertungserfolge	
Info	

Figure 1: Header from Excel Solution

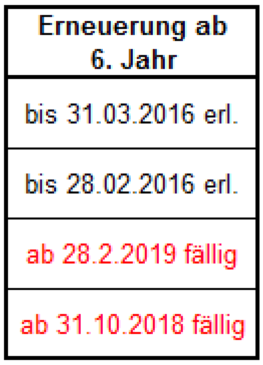
An example of such unstructured data can be found in Figure 2, which shows two columns of the active sheet and the way the information is stored. The first column contains the value yes, but this value is sometimes enhanced with a letter (which is the abbreviation of the type of evaluation the aws did), and also sometimes there is a date, which means that on this date the evaluation was ordered. The second column shows which patent-applications have been made. This column also contains information in various formats because there is no validation of this information within the current Excel solution.



Prüfung aws	Land (JKU Inhaber)
ja (12.10.2005)	AT-Pat./ PCT-Anm./ EP-Anm./ US-Pat.
ja	AT-Pat./ PCT-Anm.
ja	AT-Pat./ PCT-Anm./ US-Anm./ EP-Pat. (AT, DE, GB, FR)
ja (b)	AT-Pat.
ja (b)	AT-Pat/ PCT-Anm.
ja (c) (13.3.2015)	EP-Anm.

Figure 2: Example of data within the Excel solution

One of the main goals of invention and patent management is to adhere to all deadlines. As already shown, a patent is only valid if the fees are paid on time. A patent application can only be successfully evaluated if the deadlines for submitting information are met, and the fees are paid in accordance to the patent application type (see 2.1.1 Common process). This means, that for each patent application and granted patent all deadlines are important and need to be looked at in advance. In the current solution, the deadlines are calculated by the user and filled into various cells. The cells contain information about the timeframe the fee was paid or when the next one is due. To help the user to see the due dates the cell content is colored red (which needs to be done by the user manually). An example of such a column is shown in Figure 3.



Erneuerung ab 6. Jahr
bis 31.03.2016 erl.
bis 28.02.2016 erl.
ab 28.2.2019 fällig
ab 31.10.2018 fällig

Figure 3: Example of deadline data within the Excel-Solution

To help with the due dates it is possible to use a separate calendar solution, e.g. within Microsoft Outlook. Each deadline has to be created as a reminder. The program will then notify the user ahead of the deadlines. This solution has one significant disadvantage: the user has to maintain the deadlines on two places. Another disadvantage is, this solution is not multi-user-capable. For example, if the main user is on vacation, someone else has to use his Outlook-Account to get access to the reminders.

The last column of the Excel sheet is the “info” column, a column where the users add information about activities and events based on the invention, any patent application or any patent. This information is yet again unstructured. As shown later, the visible size of the sheet should be printable. Therefore, the information field cannot grow endlessly, and the user removes information, which is not needed anymore. This leads to a loss – even if it is intentionally – because it will be removed to have less information on the printout.

The Excel file is stored on a file server and can be accessed from people with the corresponding access rights. However, Microsoft Excel sheets are not multi-user capable as explained beforehand. To ease the access to the information, print outs are handed out to a variety of people at the JKU.

There are also advantages of the Excel-Solution:

- Changes can be made very easy, e.g. just add a new column and enter the data
- Microsoft Excel is a common known program which makes it not too difficult to use
- No experts are necessary, because the average user has the knowledge to modify a sheet
- Microsoft Excel or compatible programs are available for different computing platforms

When creating the requirements for a software based solution like the one implemented with this thesis the Microsoft Excel solution contained already a lot of information.

2.3 COTS-Solutions

Commercial off-the-shelf solutions are available for different businesses and business-processes, also for managing inventions and the patent process. At the beginning the users still have to formulate requirements and describe the processes they want to implement with the software solution. With this information the market can be searched and possible solutions could be evaluated against the requirements. [14]

On the one hand basic requirements are met by most of the market participants, on the other hand sophisticated requirements or local requirements are not met perfectly and the user has to make a trade off. Advantages of COTS-Solutions are typically: [14]

- The costs of the software are less compared to the costs of individually created software
- The solutions can be implemented faster, because the software is already finished
- Additional knowledge, which is implemented within the COTS solution, is gathered and can be used in the company.

The JKU received various offers regarding COTS solutions, however, the prices were too high and didn’t fit within the budged restrictions. This led to the idea of creating an individual implementation of a software for managing the inventions and patent processes.

Although the scope of this thesis is not to do a market research about COTS for patent management, we did a research with a Google search ("patent management software") and various products were offered. We present two of them which meet the requirements posed by the JKU department of research support. All the information about those two companies was taken from their website.

2.3.1 ANAQUA

ANAQUA offers a solution portfolio for companies or for law firms. This distinction is necessary, because a company manages the inventions for itself and is in contact with law firms. Law firms on the other side do not have own inventions, they offer their services to clients who need help in the patent process.

The software solution is module-based and the customer can buy the software to install it on his own infrastructure or use a cloud-based installation. All stakeholders can access a central repository with all information about the inventions. Workflow-capabilities allow the seamless activity tracking and scheduling of activities within the software. Costs like fees and invoices can be tracked within the solution, and the possible sale of rights and revenue creation can also be managed within the software. [15]

The solution provider offers his services in various languages, and has about 600 clients. The software solution is not only usable on inventions and patents, but covers the whole intellectual property management (e.g. trademarks). [16]

2.3.2 IPfolio

The software solution from IPfolio is web based, which means it is only used from within a browser. They sell the product only cloud-based, which means that all data is stored somewhere else. The solution is not addressed to law firms. The primary target are companies and organizations which are having inventions and need a solution to manage the rights of their patents.

It is possible to manage the complete patent process from the invention to the revenue generation within the solution. All stakeholders can participate as users in the software. With the additional module "inventors portal" the scientists can add new inventions themselves and with the integrated workflow possibilities review processes and feedback loops can be established.

The solution is customizable within the user interface of the solution, e.g. custom fields can be added. Extensive reporting possibilities and the creation of custom reports are available. As the name already suggests, this solution also does not only manage inventions and patens, but also e.g. trademarks. [17]

3 Solution overview

3.1 Development process

The implementation and introduction of an individually implemented software solution typically follows a phased process model. One such model is the OEP (oose Engineering Process), which is a practical process model for the implementation of an IT project in the agile way. [18]

The phases of the OEP are the following: [18]

- *Preparation phase*: rough planning, defining of content and goals
- *Definition phase* (design and architecture phase): analyzation of the use cases, requirements elicitation, definition of the base architecture
- *Construction phase*: incremental implementation of subsystems and integration to a product
- *Introduction phase*: test, acceptance, delivery and installation, end user training
- *Operation phase*: maintenance, backup, restore

The final software solution has been created with those phases in mind and an agile process. The agile process focuses on short release cycles, changes of requirements are also on later phases possible. The implementation is done in short incremental steps, and domain experts and developers are working close together. [19]

The *preparation* was done before starting the thesis, as the department of research support created the requirements document and the JKU searched for developer to create the software solution.

The *definition phase* started parallel with this thesis. Although the requirements were in written form, there was still not everything defined. The agile process now allows nevertheless to start development and to create a first prototype.

The following phases are now described in more detail with this thesis (construction phase is chapter 3 and chapter 4), and the remaining phases are addressed in the appendix.

3.2 Requirements elicitation

"Requirements are the basis for every project, defining what the stakeholders – users, customers, suppliers, developers, businesses – in a potential new system need from it, and also what the system must do in order to satisfy that need." [20]

At start of the individual software implementation project the requirements were posed from the department of research support. Those requirements are based on their work with the Microsoft Excel solution (see 2.2 JKU Excel solution), the known limitations (aim is to overcome those limitations) and enhancements which came to mind when writing all requirements. The result of this process was a requirement document (see Appendix) from the future users of the software. The document was structured in four parts, a short description of the content of each part can be found in Table 2.

Part	Description
Wish list	Contains requirements which would be helpful if implemented, but are not necessarily
Object / field list	From the view of the future user all objects (not in an object oriented way, more like domain objects) are listed and for each object the necessary fields (in the user interface) to store data are described. The type of data which will be stored in each field is specified, and when available default values are defined. There are various fields with reference lists. Those lists should be maintainable by the final users.
Views of the data / reports	Different views of the data (tables with various contents) and reports are defined.
Notifications	Based on various date fields and deadlines notifications are specified. Those notifications should be send by e-mail to predefined e-mail-addresses.

Table 2: Content overview of the requirements document

Additional requirements came up on the first presentation of the preconditions:

- Multi-user-capability should be implemented.
- Various computing platforms should be usable as a client.
- The solution should be run on the JKU campus and the data should be stored secure and backed up regularly.
- Easy maintenance should be possible, as the developer of the program does not maintain the solution after final delivery and acceptance.

3.3 Definition of base architecture

"Architecture is concerned with the selection of architectural elements, their interactions, and the constraints on those elements and their interactions necessary to provide a framework in which to satisfy the requirements and serve as a basis for the design." [21]

With the requirements in mind the first step in creating the software solution is to define a base architecture. This does not necessarily include the chosen technologies, however, we did the major decisions on the architecture already with the technologies in mind.

Not all requirements drive the architecture. The requirements from the department of research support describe at first an application to manage data, and an application that can send notifications on time. The first application will have a user interface, the later one does not need a separate user interface. The requirements listed in Table 3 are used as a base of the architectural process.

Requirement	Description
Multi-user capability	The software solution should allow the concurrent use from different users at the same time. Each user should have his own account. If one user edits an entity, other users should not be able to edit the same entity simultaneously. The entity in this case should match a single user interface form.
Data storage	The data should be stored within the application and the user should not be obliged to store, load or backup the data.
Usability	A user with common computer skills should be able to use the software after a short introduction of the common elements.
Multi-platform	The software solution should be installable and runnable on different computing platforms: MS Windows, Mac-OS and Linux-platforms. If the software is a client – server software, than the server software needs to be able to be hosted on the JKU infrastructure. The installation of clients should be automated (e.g. with a central software installation repository and management).
Backup / Restore	The state of the software solution should be backed up regularly. A predefined backup and restore process should be established and the application should be able to start from a previous stored backup.
Security	The software solution should only be accessible by authenticated users. If the solution exchanges data somehow, this data should be encrypted with industry standardized encryptions.

Costs / Licenses	The software solution has to use royalty free software (no license costs), ideally open-source and public available components.
Maintenance	The software solution should be maintained in the future by a developer with programming skills in common programming languages and technologies.

Table 3: List of requirements which drive the architecture of our solution

A modern definition of software architecture is given by Kruchten, and shows the importance of decisions within the architectural process:

> *“An architecture is the set of significant decisions about the organization of a software system, the selection of structural elements and their interfaces by which the system is composed, together with their behavior as specified in the collaborations among those elements, the composition of these elements into progressively larger subsystems, and the architectural style that guides this organization - these elements and their interfaces, their collaborations, and their composition.” [22]*

Before we can decide upon the architecture, we need to find possible solutions. They are intertwined with the various requirements stated in Table 3 and also intertwined with other options. This means that not every option can be seen isolated, because they may depend on each other, and the following sections are not meant to be in exact sequence.

We do use common patterns for our solutions. A pattern is a commonly documented and reusable solution to a common problem, e.g. the MVC-pattern for splitting the user interface into distinct roles. [23] A pattern is not specific to a programming language or technology. It can be seen as a modus operandi to solve a problem, and has to be implemented in the environment where it will be used.

3.3.1 Multi-user capability

The system should allow the usage of the solution from multiple users concurrently, as well as sequentially. The users should be able to access the same data concurrently on different computers. As we will show in the data storage, a DBMS as central data store will allow concurrent access to the data.

Although a DBMS allows the concurrent access, when writing the same data potentially problems may occur. The user who saves later will overwrite the changes from the user who saved before him, even if he does not know that other data changes have been made. This can be circumvented if the data contains a version, and before updating data the version is compared with the version the actual data has, and only if the versions are equal the data will be saved.

This version mechanism has one disadvantage: how should the user interface, and later on the user, react to a possible inequality of the versions? The user interface could display a message as information to the user. However, should the user interface display the data with the newer version from the central data store, or should it keep the entered information of the user and allow him to forcefully save the changes (again overwriting other changes)?

Another possible option is to use a custom locking mechanism. When entering the edit state of an entity (typically one user interface form) the data in the central store is updated with the information of the lock, the user who owns the lock, and date and time when the lock was created. Each approach to edit data (an entity) will upon allowing the edit again check the lock-information. If there is a lock, then the user cannot access the edit form. After successfully editing the entity the user updates the data (or just discards his changes) and the custom lock is removed. This solution has one issue: the user has to remove the lock with at least one action (save or discard). If the user just closes the application, or the computer hangs, the lock will be there until someone removes it by hand. However, a solution to this issue is to check on edit-request if the user is the same (the same user can always edit the entity he has a lock on), or if the user is different if a certain time has passed (e.g. 30 minutes), allowing another user to edit the data. The information about the lock can be displayed in the user interface, showing other users before they try to edit that they cannot edit.

After discussing this issue with the project team we decided to *use the custom locking* mechanism.

3.3.2 Data storage

The requirements state that the data should be stored within the solution. There are different kinds of data storage available. We could store the data *locally* on the client, or *centrally* on a server. However, if we want to address *multi-user-capability*, the local storage may pose some problems. The data has to be transmitted to each user, and the data has to be synchronized. On the other side, if we use a central storage, and each user gets the data which is currently needed, the multi-user-capability may be easier to add.

The type of the storage can be a file (handled by the software solution) or a database. The database can be installed on a central location, the data can be kept within the database, and the DBMS will allow structured access to the data. Concurrent use of the data is one of the features of the DBMS. On the other side, the custom-made file solution will have to provide features like concurrent access, safe operations and not losing data. The DBMS fulfills those requirements if it handles the ACID properties. Another aspect is the maintenance and further development of the software solution. With the custom file solution, every new developer has to learn the usage and cannot use already gathered knowledge. However, when using the DBMS a developer who has once worked with a DBMS already has the knowledge on this part of the software solution.

Additional to the already shown options, advantages and disadvantages the risk of a custom made storage solution would be bigger than the using of an already available technology. As there are no requirements prohibiting the use of a DBMS the *chosen option is to use a DBMS* which is *installed on a central server*.

By using a central server another architectural decision is immanent: we will *use the client-server* paradigm. The server offers services, which are consumed by clients. The services and the clients have to use the same protocol. However, the paradigm does not specify that there is only one server, there could be multiple servers, and services on a server could also be clients from another service. [24]

3.3.3 Multi-platform

The requirements state, that the software solution should be able to be used on different computing platforms. As we already decided to use the client-server paradigm, we have to distinguish between the clients and the servers.

The clients are used to look at the data or modify the data. The clients will be used to display a user interface, interact with the user, and communicate with the services from the servers. There are various solutions for multi-platform user interfaces, e.g. Qt or Java. We want to address the maintenance requirement, therefore we favor a solution with a widely accepted and used technology. As such Java is a preferable solution. However, a Java based application has to be installed on the clients, and the clients also need a Java Runtime (which could be bundled into the installation).

Another option is to use the web browser as client. The big advantage is that there is no installation necessary (except if some plugin would be required), the address of the web server is the starting point of the application (from the user's point of view). The common disadvantage of web based applications is browser compatibility, especially for older or future browsers. To address older browsers the developer has to specify stylesheets and client scripts to work also on older browsers. To address future browsers the developer has to comply with the current standards, and allow easy manipulation of the source code of the client application.

Because on all the different computing platforms in use at the JKU a modern graphical web browser is available the *browser is chosen as client-environment*. With the rise of HTML5 in modern browsers there is no requirement for browser plugins.

The decision upon client-server-paradigm and using a browser as client leads to the follow up decision on the server. As already stated the *programing language should be Java*, because future maintenance should be possible with using a common language. There are multiple options for creating a web-based application based on Java, the server options (based on standard technologies from Java) are described in Table 4. Although the options not all produce a client user interface, all options are running on the web server.

Option	Description	UI-Implementation	Content transported between client and server
Servlets	Servlets are Java classes which extend the capabilities of the server. They allow to intercept the request-response mechanism and are able to produce HTML. [25]	Server-side, client enhanced, coupled	Plain HTML, HTML-form-data (on submit)
JSP	This is a Java view technology, where the application code runs on the server. The request-response mechanism allows to output data, HTML or JavaScript dynamically. [26] The extension of the server to be able to work with JSP is done by servlets.	Server-side, client enhanced, coupled	Plain HTML, HTML-form-data (on submit)
JSF	JSF is a component based Java framework for building web applications. The Model-View-Controller pattern (MVC) is integrated into the framework. [27] JSF is an addition to JSP.	Server-side, client enhanced, coupled	Plain HTML, HTML-form-data (on submit)
REST	This is a kind of programming paradigm, where between the client and server only data is exchanged (without markup for the user interface). The implementation is either via servlets or by using a framework which provides the necessary functionality.	Client-side, decoupled	Just data (encapsulated with JSON or XML)

Table 4: Overview of Java web server technologies

Depending on the option we have to use different client-implementations. If we use an option with server-side user interface implementation, we have to send to the server requests from the client. Those requests may contain additional data, e.g. when the user submits a form. The server accepts those requests and handles them with the respective servlet (or JSP/JSF). Afterwards a response is created. This response can be dynamically created. However, the response is a mixture of HTML markup for the user interface and data which needs to be displayed.

On the other hand REST allows a clear separation of markup and data, because the server sends only data to the client and the client also sends only data to the server. This option allows also the usage of the data within other clients (e.g. a Java program, or another web-application), whereas the other options have to be changed for other consumers. Another advantage is the testability, because REST does not contain UI-code (markup etc.) it can easily be tested. There are more advantages of using such a decoupled architecture, however, those did not drive our decision.

The different options could also be used in a parallel way. Before deciding which option may fit best we need to have a look at the possible client-solutions within the browser. Although most of the client-solutions can be created with every server-solution, the best fit approach is chosen based on using the server-option the way it was designed for. Table 5 shows an overview of the customized options we found for using the application, and Figure 4 shows how the options evolved over time.

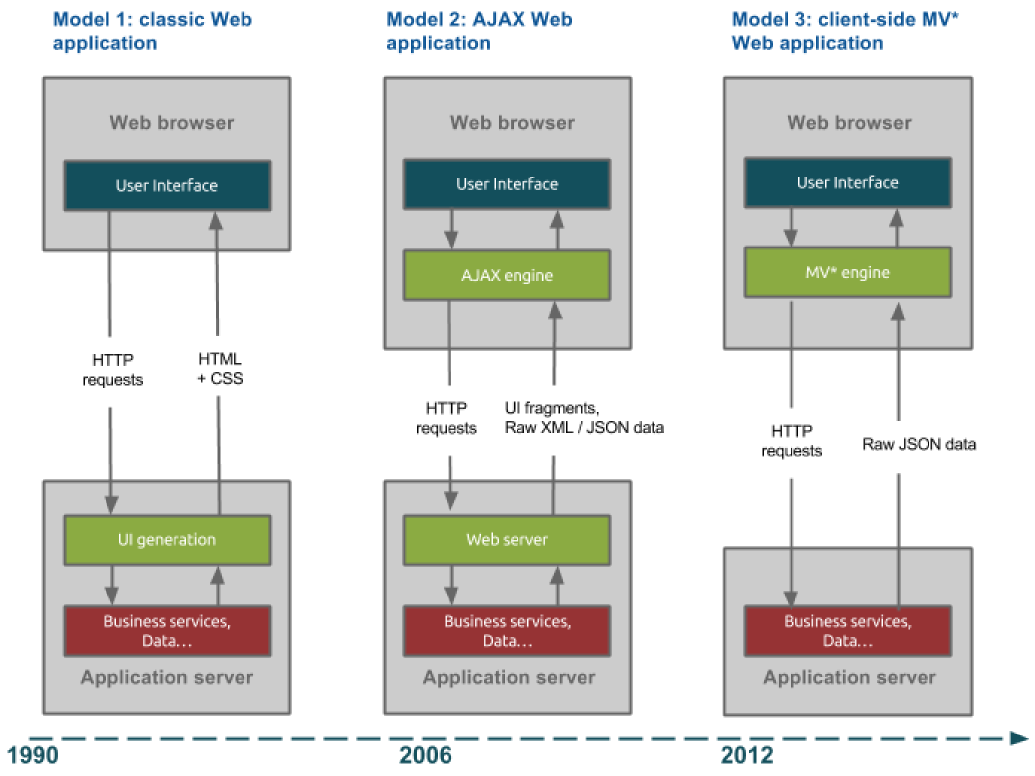

Model 1: classic Web application
Model 2: AJAX Web application
Model 3: client-side MV* Web application
Web browser
User Interface
AJAX engine
MV* engine
HTTP requests
HTML + CSS
UI fragments, Raw XML / JSON data
Raw JSON data
UI generation
Web server
Business services, Data...
Application server
1990
2006
2012

Figure 4: Timeline of different client-models [28]

Although Figure 4 shows no transport of UI-elements with models 2 and 3, this depiction shows only the dynamic part. These models still need a web-server which provides the static elements. This could be the same server which provides the dynamic content, however, it could also be another server. The model 3 introduced the term “MV* engine”, which will be described later. However, in the phase of defining the base architecture SPA is used as a common term to describe the client side of model 3.

Option	Description	Fits to server option
HTML	A pure HTML solution. This would not use any client-side JavaScript and means that there are no client-side dynamic changes possible. This also means a pure page-based request-response mechanism.	Servlets JSP JSF
HTML + JavaScript	With the addition of client-side JavaScript dynamic user interface features can be implemented which are executed on the client. The communication with the server is still based on a page-based request-response mechanism.	Servlets JSP JSF
HTML + AJAX	With AJAX it is possible, to communicate with the server outside of the normal page-based request-response mechanism. The client can for example load data from the server.	Servlets JSP JSF REST
SPA	Single Page Applications are also based on the AJAX principle. However, the user interface is build more like in a classic installable application and does not use paradigms like a page.	Servlets REST

Table 5: Overview of client-side options

With the usability requirements in mind, the advantage of decoupling the server from the client and the rise of SPA as major paradigm for web-applications led to the decision, to *use REST on the server-side and SPA on the client-side*.

One of the biggest advantages of this decision is, that we can independently develop and release the client and server. We could even change the technology or framework used to implement either one, as long as we use the same contracts, which are formed by our REST interface. For the implementation of the client we will use HTML 5. [29]

3.3.4 Usability

To increase the usability of the software solution the user interface should be intuitive and self-describing. A user with the domain knowledge should be able to work with the software solution after a short introduction. We use the browser as a client-environment, we can use common user interfaces of web-applications or web-sites, which the user may already know.

The user interface should use little colors to allow the focus on the content and to not distract the user, and it should be possible to increase or decrease the user interface with the browser to allow an adjustment of the size if needed (e.g. for visually impaired user). The navigation within the software solution should be based on a menu, where the user chooses from the available options. Buttons should be used to save data, when deleting data confirmation dialogs should be used. All modification actions should display the result as a success-message or an error-message. If there are issues with the data or modification process, hints should also be displayed guiding the user to the solution.

Today various UI components are freely available, Bootstrap being one of the most common one. Bootstrap is a HTML, CSS and JS framework for the development of web-projects. It provides UI-Components which have been designed with usability in mind. When using those components we can provide a uniform styling and the same UI metaphors across our solution. [30]

The posed requirements can be met with *using a SPA, using Bootstrap* and the careful selection of colors for the user interface.

3.3.5 Backup and Restore

With the already chosen options we have to check if they also meet the requirements for backup and restore. With using a DBMS we store the data within the DBMS. We have to choose a DBMS where we can backup and restore a database (containing the data).

On the web- and application-server we do not store data. However, the log-files produced by the application may be of interest. They will reside on the file system and can be easily copied to a central backup-device.

The client-side does not store any data, therefore we do not need to do any backup of our application on the client-side.

3.3.6 Security

The security requirements are primarily to protect the data from unauthorized access and from losing the data. The latter issue was already addressed in a previous section (see 3.3.5 Backup and Restore).

Authentication is done by using a login mechanism. Only authenticated users should be able to gain access to the data (and software solution). The JKU provides a central authentication authority by using Shibboleth. [2] The software solution should use this service. The main advantage is, that the employees of the JKU can use their common login of web-applications (Single Sign On).

Authorization is also done by the central Shibboleth authority. The authorization is based on roles, and a user can have multiple roles assigned to. The individual features of the user interface are based on the roles, and the server has to check if the user is authorized to use a feature (process data).

For authentication and authorization we *use the central Shibboleth Service* of the JKU, and check the credentials client-side as well as server-side.

Additionally the data exchange between client and server is also in need of protection. Because we already decided upon using a web-application approach, we can *use TLS* to encrypt the traffic. [31] Whatever other decisions we take, we always have to make sure that the solution can be used with TLS.

3.3.7 Costs / Licenses

The decisions already made pose to use the following architectural elements:

- DBMS for holding the data
- Web-server for providing a REST service
- SPA for providing the client-side user interface and communication with the REST service
- TLS for the encryption of the communication between server and client

The requirements state, that we should use only solutions with no license costs, ideally open-source. When choosing appropriate solutions for each element, we still have to keep in mind some of our requirements, mainly the requested platform independence of the solution.

Beside the requirements and the variety of possible solutions the know-how of the developer and experience with previous projects can also restrict or promote the decisions. However, the not existing know-how of an option should not automatically exclude that option. Another driver of our decisions is the fact, that the software solution will be installed on an environment which is then maintained by the staff of the department of IT services of the JKU.

There are multiple DBMS available, some of them without license costs and even open-source. We decided upon *using PostgreSQL*, because it is available on different platforms and in previous projects all requirements where met with the product.

The options for the web-server (and web-application-server) are also plentiful. The know-how and the future environment of the software solution leads us to *use Apache Tomcat*. The product has been available for a long time and there is an active development community available, which creates new versions. Tomcat allows the usage of the Java-Server-Technologies (see Table 4) and can be installed on different platforms.

A SPA could be developed by ourselves. However, we prefer to use a framework which provides a rich user-experience out of the box. There are many frameworks available. All of the SPA frameworks will meet the requirements of usability and will work with a REST service on the server. They are open source and can be used without license costs. One of the decision driver could be the availability of information about the framework, and the future the framework has, which mostly depends on the developer of the framework. The two most prominent ones are *React.js*, which is developed by Facebook, and *Angular.js* (Angular), which is developed by Google. Both are under continuous development, which means that new versions will be published, and the support of the two companies behind the frameworks is active. With both SPAs we could implement our software solution. Because we already know how to use Angular we decided *to use Angular*. This decision was also tested with a prototype to show the project team a usable user interface.

We do not need to explicitly decide upon TLS. TLS will be used, and the exact version or issuer of the certificates is not of interest in this thesis.

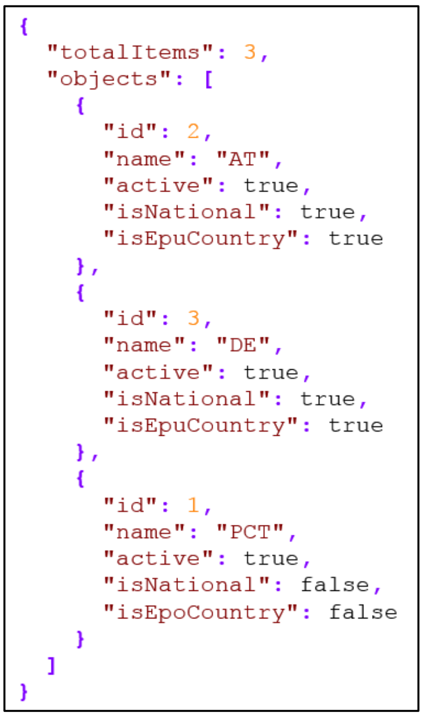
3.3.8 Base architecture

With all decisions so far we have created a base architecture of our solution. We decided upon the client-server architecture. On the client side we will use Angular as SPA-Framework. On the server side we will use Tomcat as application server, and PostgreSQL as database for storing the data of the application. Client and Server are connected via a network and use TLS to protect the transmitted data.

With the decisions so far we have also selected programming languages for developing the solution:

- The DBMS PostgreSQL is used with SQL, a language for querying DBMS.
- The application server in Tomcat is developed with Java, a modern object oriented programming language.
- The client SPA is developed with JavaScript, which is a script language. However, elements of HTML and CSS will be used too.

For the exchange of our application data between the server (REST) and the client (SPA) we will use JSON, which can be used as data-interchange format. It is written in clear-text, which means that humans can read it. However, because it is an interchange format machines can also easily read JSON. Figure 5 shows an example of the JSON which is sent from the server to the client upon requesting patent application countries.



```json
{
  "totalItems": 3,
  "objects": [
    {
      "id": 2,
      "name": "AT",
      "active": true,
      "isNational": true,
      "isEpuCountry": true
    },
    {
      "id": 3,
      "name": "DE",
      "active": true,
      "isNational": true,
      "isEpuCountry": true
    },
    {
      "id": 1,
      "name": "PCT",
      "active": true,
      "isNational": false,
      "isEpoCountry": false
    }
  ]
}
```

Figure 5: Example of JSON - the patent application countries

4 Implementation of Invadmin

The product implemented is named invadmin (invention administration). We have already described the usage of the client-server pattern. Figure 6 shows the base architecture of the web application.

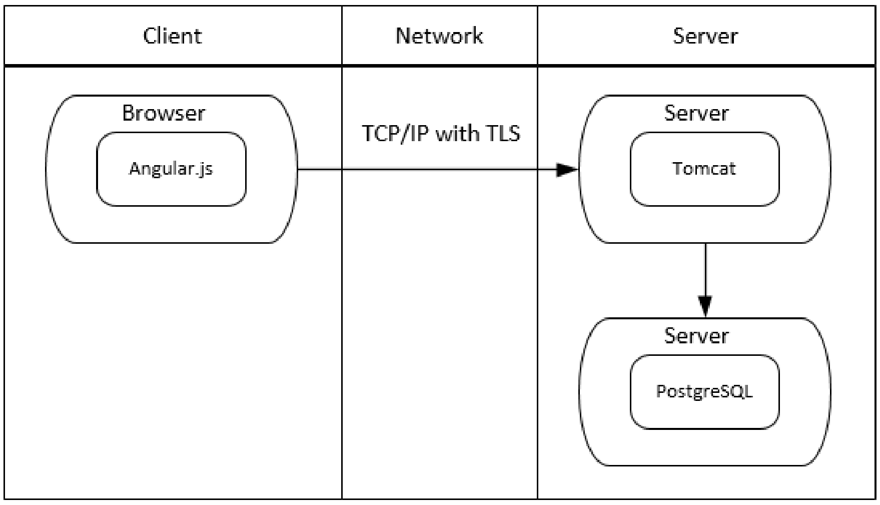


Figure 6: Base architecture of our software solution

This architecture can also be referred to as multi-tier architecture. *"A Multi-tier Architecture is a software architecture in which different software components, organized in tiers (layers), provide dedicated functionality."* [32] With the repeating use of the client server paradigm we get the multi-tier architecture. The server of one component may be the client of another component. One of the advantages of the multi-tier architecture is the individual deployment possibilities of each tier. We can install the PostgreSQL on a completely different server than the Tomcat, even different operation systems are possible. This offers more installation options in the future.

4.1 Application Server

In Java there is a distinction between web container and application server. The latter one has also support for EJB and transaction management among others. The distinction between the different servers is the following:

- Web server: serves static content.
- Web container: serves dynamic content, like JSP or web services.
- Application server: web container plus enterprise features like distributed transaction management, object pooling.

Tomcat is known as a web container, because it does not provide the enterprise features. However, Tomcat is also a web server, which means it can also serve the static content. Although it is strictly speaking a web container, we further use the term application server in this thesis for easier understanding.

4.1.1 Overview

We use a layered approach for our application. The layers are used as functional borders within the application. Each layer has his own purpose. In Figure 7 we show the layers used in our application server.

The router layer accepts the requests from the client and decides the individual method in the service layer based on rules. The service layer does the concrete work (e.g. getting data from the persistence layer as domain model objects, and transforming objects to the JSON representation). The persistence layer handles connections to the database and loads data from the database or stores data to the database.

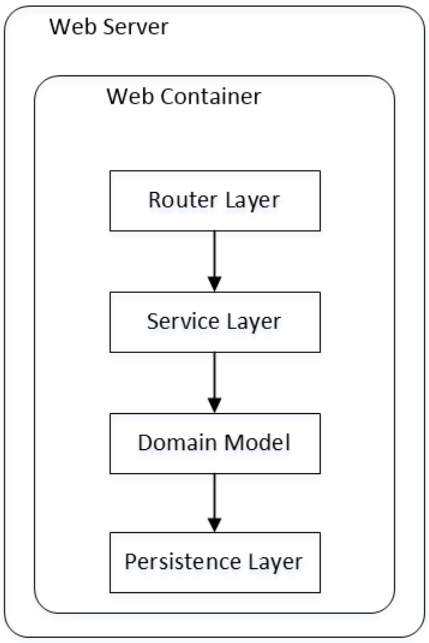


Figure 7: Application Server Layers

Although we use a layered approach, we also have additional elements, which are part of the layers and in the following sections described in more detail.

4.1.2 Domain Model

A domain model is *"… an object model of the domain that incorporates both behavior and data"*. [23] Fowler also distinguishes between two styles of domain models, a simple, which basically maps one database table to one domain object, and a rich, which *"… can look different from the database design, with inheritance, strategies, and other [Gang of Four] patterns, and complex webs of small interconnected objects."* [23] We use inheritance in our domain model, however, the other parts of the definition do not meet our design. Therefore we have designed a *simple domain model*.

The domain model is part of the web-application, although in our application is partly also available on the client side, because we transform our domain model objects into JSON, which we send and receive from the client.

The domain model reflects the requirements to store the data. The application allows the storage of e.g. an application country. This application country is an instance of the domain object “ApplicationCountry”, where the properties of this instance are the data fields of an application country. This relationship can be seen in Figure 8.

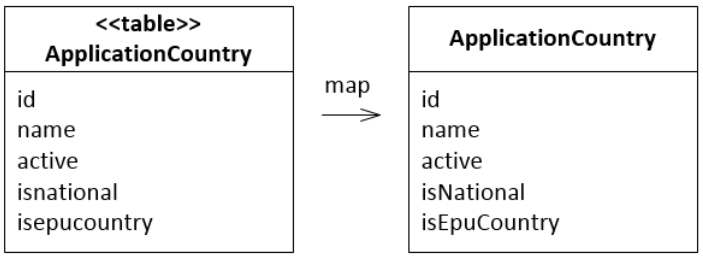


Figure 8: Table - domain object relation

However, not all domain objects are equal to their database equivalent. Some domain objects have additional properties, where the service layer gets the data for this fields from other database tables. One example is the inventor. In the relational model it is a table with foreign keys to invention and person. This way a person can have multiple inventions, and an invention can have multiple inventors. The client has to get the persons and the inventors separate and join the data on the client. To circumvent this client work we create the inventor domain object, which does not only have properties from the inventor table but also properties from the person (see Figure 9). Those properties are read-only and are just used to send JSON to the client which contains already the necessary information.

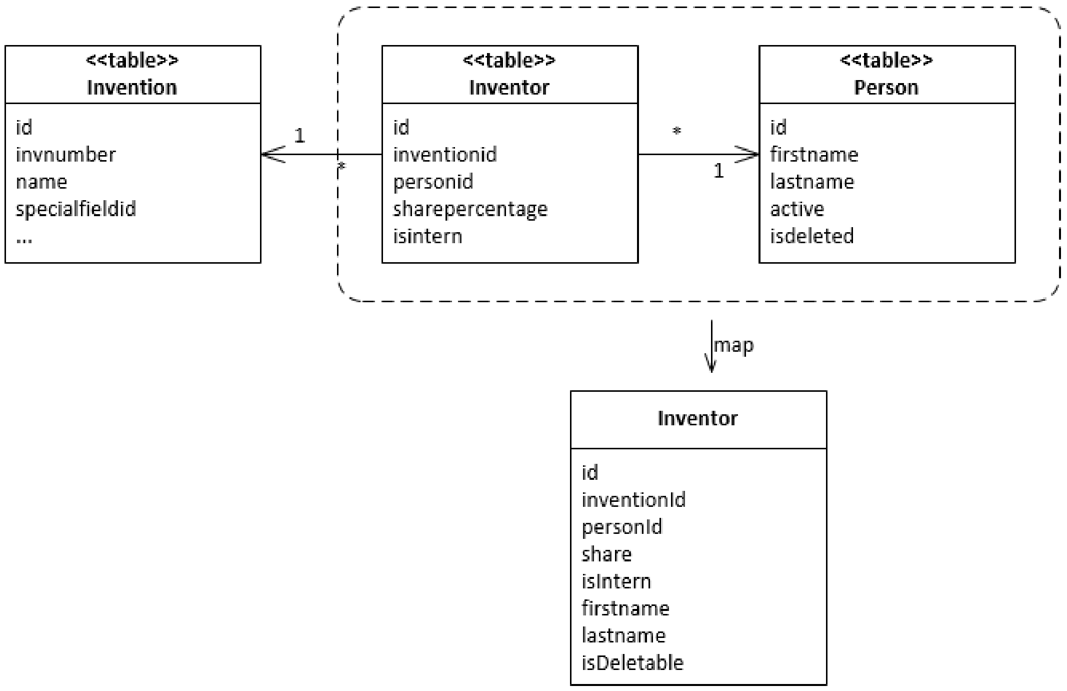


Figure 9: Multiple Tables – one domain object relation

There is also the property “isDeletable” (see Figure 9), which does not show up in the relational schema of the database at all. During the demonstration phase of the prototypes the requirement to delete inventors arose. Typically this is needed if there was some erroneous input done. However, if the inventor is already used in some other business context the deletion should not be possible. There is currently one other context, namely the remuneration of inventors, where the inventor-entity is used again. On creation of the inventor-objects we already evaluate if there was remuneration, which will not allow the deletion. The user interface can therefore remove the delete action on displaying the inventors, which increases usability. Otherwise we still have to check if the deletion is possible, and if it is not we have to display the user an error message, stating that he is not allowed to delete that entity.

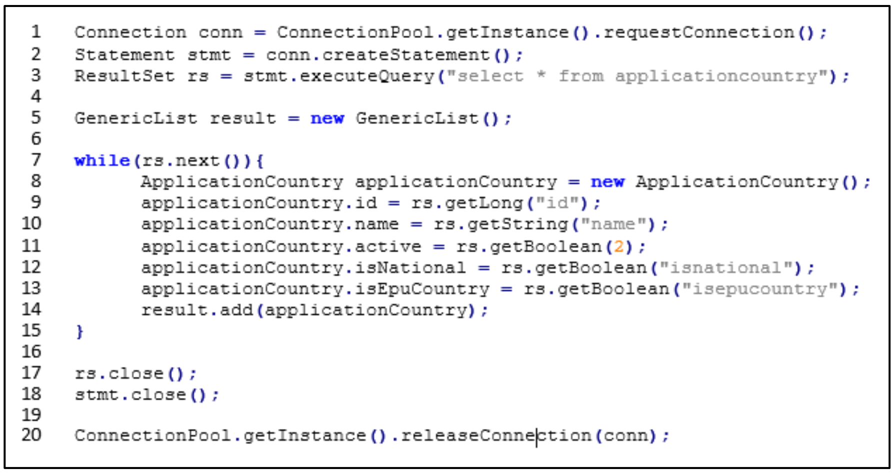
Because we use a relational database to store our data we have primary keys for the individual database entities. We use numbers as primary key (fieldname is id). This primary key can be seen as identity, because it allows the distinction between objects of the same type, and it identifies the individual entity. This pattern is also described in Fowlers work with *“Saves a database ID field in an object to maintain identity between an in-memory object and a database row”*. [23]

4.1.3 Persistence

To access the database and retrieve data we need to use an API. In Java applications this is typically JDBC. There are various specific properties necessary to connect to a database: the host address where the database can be found, the name of the database and authentication details (user and password). It may be that special JDBC-drivers are necessary, this depends on the vendor of the DBMS.

JDBC allows uniform access to the database. When writing code based on the JDBC-API we can access different databases with the same code, however, often databases still have individual differences from other databases (e.g. is the LIMIT clause of SQL not standardized although it can be used in PostgreSQL and MySQL for example).

In Figure 10 we present a code snippet as an example of database access with the JDBC-API. We fetch the application countries from the database (Exception handling is omitted due to better readability, and a connection pool is used instead of the individual code to create and close a database connection). When looking at the presented code, we may find some potential problems for the future. We use the datatype string in various lines (3, 9, 10, 12 and 13). We use the datatype integer on one line (11). This line could also be of the form like the previous ones, however we wanted to show the potential problems which may arise.



```
1   Connection conn = ConnectionPool.getInstance().requestConnection();
2   Statement stmt = conn.createStatement();
3   ResultSet rs = stmt.executeQuery("select * from applicationcountry");
4
5   GenericList result = new GenericList();
6
7   while(rs.next()){
8         ApplicationCountry applicationCountry = new ApplicationCountry();
9         applicationCountry.id = rs.getLong("id");
10        applicationCountry.name = rs.getString("name");
11        applicationCountry.active = rs.getBoolean(2);
12        applicationCountry.isNational = rs.getBoolean("isnational");
13        applicationCountry.isEpuCountry = rs.getBoolean("isepucountry");
14        result.add(applicationCountry);
15  }
16
17  rs.close();
18  stmt.close();
19
20  ConnectionPool.getInstance().releaseConnection(conn);
```

Figure 10: JDBC example of fetching application countries

The SQL query statement has to be correct, but because it is just written as string we do not get a compile time check. The compile time check is done during the compilation of the source code. Only valid source code can be compiled, and the compiled program can be used. The main problem, which may occur later is, that the SQL query is not valid. The two cases are due to changes in the relational model, e.g. renaming the table, or due to changing the DBMS and the other one does not allow the used SQL syntax.

The access to the data of the table fields is either with the name of the field (e.g. line 9 “id”) or with the zero based index of the ordered table fields (e.g. line 11 “2”). However, both may change in the future (either due to changes in the previous SQL query or to changes in the relational model). In the worst case we map the wrong database-field to a property (when using the index).

The last lines of the presented code snipped are also very important. We have to close the various objects. This can be easily forgotten and will lead to unused objects in memory.

One common solution to the described issues of using JDBC-API directly is an object relational mapper (ORM). The mapper will use predefined rules to map between the database (tables and fields) and the objects (properties). The ORM uses the JDBC-API, although the application developer uses the ORM instead of directly address JDBC-API.

Because we have a simple Domain model (see 4.1.2 Domain Model) we can use the Active Record pattern. This pattern is defined as *“An object, that wraps a row in a database table or view, encapsulates the database access, and adds domain logic on that data.”* [23] The object relational mapper will create active record objects. These objects have typed properties for each field, which means that the string and index issue to access fields as described above is gone. Although this pattern is defined as described, our solution uses a slightly different modification. We do not use the ORM provided classes (described below) as our domain model. When using an already available ORM-tool we have to adapt to that tool.

When getting data by id the Active Record pattern is very useful. The SQL query could be created automatically, e.g. there could be a method "*getByKey(key)*" and the object will load the record with that key from the correct table, without adding a SQL query to the source code of the application. [23] For more sophisticated queries we still have to write SQL as string into our application, and still have the same issue as using JDBC-API directly.

This leads us to the ORM-tool which allows us, next to the already described Active Record pattern, handling SQL queries in a type safe manner with full compile time check. Our decision is to *use jOOQ*, because it is available as open source, it is actively developed, and fits to our requirements. [33]

Based on the relational model, which is retrieved by querying the database, jOOQ creates multiple classes for different purposes, which are organized in Java packages and described in Table 6.

Package	Description
Schema	jOOQ allows the usage of a typed SQL-DSL within Java. This DSL needs information about the database schema used, which are created in this package. It contains the tables, various keys and sequences.
Tables	This package contains one class per table, which describes the schema of that table, and this is also used with the SQL-DSL.
POJOs	This package contains one class per table, which can be used as POJO. The classes only contain properties (one of teach table field) without any behavior attached.
Records	This package contains one class per table and also properties per table field. Additionally these classes have a method to persist the record (the behavior).

Table 6: List of jOOQ packages based on the relational model

The creation of the various classes is done by running a command line tool. This can easily be integrated into our development process. Whenever we change the relational model, we have to run the tool. This recreates all classes. The previous example with changing the table name will lead to a changed class name, and all code which used the old class name will not find the class anymore, and we will get a compile time error and can easily fix that issue. When creating the SQL queries with strings we do not have that check and will find the issue whilst testing or in deployment phase.

When using jOOQ for retrieving data from the database, the various classes may be used to create objects based on the rows from the database. In Figure 11 we present a code snippet, which retrieves all application countries (again without exception handling). Although we still have to map to our domain model, we now have no strings or index used, and as a result typed objects.

```java
List<Applicationcountry> dbApplicationCountries = ConnectionPool.getContext()
            .selectFrom(Tables.APPLICATIONCOUNTRY)
            .fetchInto(Applicationcountry.class);
```

Figure 11: jOOQ example of fetching and mapping application countries

Because that example is very straight and easy, we want to show a more complex example. As already described in Figure 9 sometimes we want to map from multiple tables to one domain object. With Figure 12 we present the loading of the data mentioned in the previous figure. Because jOOQ has the complete schema information, we could use methods like “onKey()” (line 3), without explicitly specifying the fields we want to join on (which, in contrast, we would have to make when writing plain SQL). We use typed field-names (yet again compile time check). The parameters are also typed and checked against the generated schema information, which will prohibit type errors.

```java
1   Result<?> dbResult = context.select()
2         .from(Tables.INVENTOR)
3         .join(Tables.PERSON).onKey()
4         .where(Tables.INVENTOR.INVENTIONID.eq((int) this.invention.id))
5         .and(Tables.INVENTOR.ISDELETED.eq(false))
6         .orderBy(Tables.PERSON.LASTNAME, Tables.PERSON.FIRSTNAME,
7   Tables.PERSON.ID)
8         .fetch();
9
10  for(Record record : dbResult) {
11        InventorRecord inventorRecord = record.into(Tables.INVENTOR);
12        PersonRecord personRecord = record.into(Tables.PERSON);
13        //do something with the two instances
14  }
```

Figure 12: jOOQ example of fetching and mapping from multiple tables

We now use the typed DSL-SQL and typed parameters. This will prohibit SQL-injection attacks, because we do not string concatenate at all, and the ORM-tool also uses typed parameters when accessing the database.

We do not use the generated classes as our domain model. There are at least two reasons:

- *Decoupling:* with decoupling our application from the generated classes we can easily change the ORM-tool

- *Overwriting:* with each change of the relational model we recreate all classes. If we want to add properties those will be overwritten during the next recreation-cycle. Models like presented in Figure 9 have to be created in some other way.

There is the domain model, and there are the generated classes from the ORM-tool. We now have to bring them together. This is done via mapping-methods on the domain model. We could separate the mapping-code from the domain model into individual mapper-objects, however we decided to include them in the domain-objects. When changing the ORM-tool, we either have to change the mapping-code in the domain-object or in the individual mapper-object, but nevertheless we have to change it somewhere.

The mapping is done in two directions, namely the creation of the domain-model (from the active record) and to the active record (from the domain model). The mapping is just assigning values from one object to the other.

One of the requirements states, that we have to record the creator and last changer of the entities (e.g. invention or patent application), as well as the date and time information when this was done. We have to add the necessary columns to the tables in the database, and on every operation with the data we have to set the values according to the actual context. However, it is repetitive code, and the biggest disadvantage is, that we can even forget about writing the code. When we add constraints on the data in the database, we will get an error from the database if we have forgotten to set this additional data. However, not everything can be controlled with database constraints.

One solution to automatically store the additional information (e.g. creator) is to use the ORM-tool. However, currently there is no support included in jOOQ. Because it is an open source, we can get the source code and modify it to work the way we want it. However, we decided not to change the source code. The reason is that if we change the source code, when integrating a future release of the ORM-tool the maintainer of the software solution has to integrate the changes to the source code again.

Another solution is to write the code in place of the various operations (creation and update of entities). Again, this would be massive code replication. The solution path we went was to modify the generated Active Record classes. jOOQ generator provides hooks where we can add code to the generated classes during the generation process. With this hooks we can still recreate the classes every time we change the database schema, however we do not have to change the files directly.

We created an extended JooqGenerator, which is an extension of the default generator published with jOOQ. Within this extended generator we add source code at the end of the generated Active Record classes during the generation process (because of the hook provided).

The generated classes now contain a method “persist(int userId)”, which can be used from the different operations. This method will look at the schema of the table, and if the schema contains the additional information columns, the values will be set. There is still the method “store()”, which also persists the active record. However, when using that method the additional information (e.g. creator) is not stored in the database. Constraints on the database will reject the data if the creator is not present (when creating new entities). The update of entities will still be possible. The “store()” method can only be removed with the change of the ORM Tool.

4.1.4 REST

In the previous sections the persistence layer and the model layer were described. They are the base layers for the service layer. In the presented application this layer is modeled for the REST resources.

In the service layer the application receives requests to reply with data, or receives data to store or delete. All these are operations on the domain model and persisted using the database.

REST is an architectural style, which uses HTTP methods for the communication between the client and the server. The concept of resources is used in this style. A resource can be seen as a thing which is uniquely identifiable. This is typically done by assigning a unique URI to the resource. The URI consists of the HTTP-address of our server, the port where the server listens, and the path of the resource on our server. [34]

HTTP defines *request methods*, which can be used on sending a request to the server. [35] Because REST uses HTTP those methods are also used in REST and they have the same meaning like in HTTP. This way we have only one URI for a resource, however we are able to make different operations with this resource. [34] Table 7 shows the commonly used HTTP methods and describes what they do with a resource on the example of application countries.

Method	URI-path	Description
GET	/applicationcountries	Gets the list of all application countries.
POST	/applicationcountries	Creates a new application country resource. In our application this means that we create a domain object “ApplicationCountry” and store this object in the database. The result of the POST operation is that the client receives the unique identity of the object.
GET	/applicationcountries/5	Gets the application country with the unique identifier 5 assigned.
PUT	/applicationcountries/5	The application country with the unique identifier 5 assigned will be

		updated with the data from the request.
DELETE	/applicationcountries/5	The application country with unique identifier 5 assigned will be deleted.

Table 7: List of HTTP request methods and their REST meaning

One of the rules of HTTP-request methods is, that they should work *idempotent*. This means that the outcome of an operation is the same, even if it is applied multiple times. [36] The GET, PUT and DELETE are specified as idempotent. [37] Although they are specified in that way, it is possible that the server does not comply with that. This may be a problem if the clients of the service expect an idempotent method. [36] Our solution uses the methods idempotent, however, multiple DELETE requests on the same resource will not get the same response. The first request will get the response with HTTP-code 200 Ok, the remaining requests will get the response with HTTP-code 404 not found. Nevertheless, idempotence in the specification is defined as side-effect free, and multiple requests to delete a resource are side-effect free.

In our application we only use the request methods described in Table 7. REST uses not only the request methods from the HTTP, but also the *HTTP-codes* for the response. Each response from a web-server has an HTTP-code associated. [35] Those codes can be used to show the content, an error message or to send another HTTP-request to another URI.

The most used HTTP-codes in our application are listed in Table 8 and a description when this HTTP-code is used is given.

HTTP-Code	Name	Description
200	Ok	The request was ok and the requested operation was done. The response may contain additional data (e.g. the requested resource).
201	Created	This code is used when the HTTP-verb POST was used and the resource was successfully created.
204	No content	If the requested resource is available, but the resource does not have any data, this code is used. This code may arise when the request for a list of resources is done, and there are no single elements at the moment. Then the list is empty, which is “no content”. This code should not be mixed up with the code 404, which means that there is no resource at all.
401	Unauthorized	If the requested resource is only available to users with specific roles, and the request is done without

		credentials or with credentials which have no access to the resource this code is sent *without* the resource.
404	Not found	The requested resource cannot be found. This is either because it was removed or the URI-path is wrong.
406	Not acceptable	The request is processed and the resource may exist (or may be created). Nevertheless, the server does not accept the request as expected. This is e.g. the case when not all data for a new resource was presented. A response with this code is done when conditions are evaluated and the response does not fit. The response will contain additional data about the error.
500	Server error	The request is processed and the resource may exist (or may be created). However, an unforeseen error occurred. This could be a bug in the application, or e.g. an occurred exception. In those cases the response will contain additional data about the error.

Table 8: List of HTTP-codes used in REST

Another concept of HTTP – *stateless* – is also used in REST. This means that the client always has to send all data to the server, which is necessary for the server to execute the corresponding method. There should be no state held or created for individual clients. This allows the distribution of the service to multiple servers without maintaining a relation between individual servers and clients.

A Java application can be enhanced with RESTful services with the JAX-RS. [38] This API uses annotations on classes and methods to implement RESTful services in Java. The reference implementation for the JAX-RS is *Jersey, a RESTful Web Services framework*. [39] The framework is open source, and adds additional functionality not defined in the JAX-RS. We use this framework in our software solution.

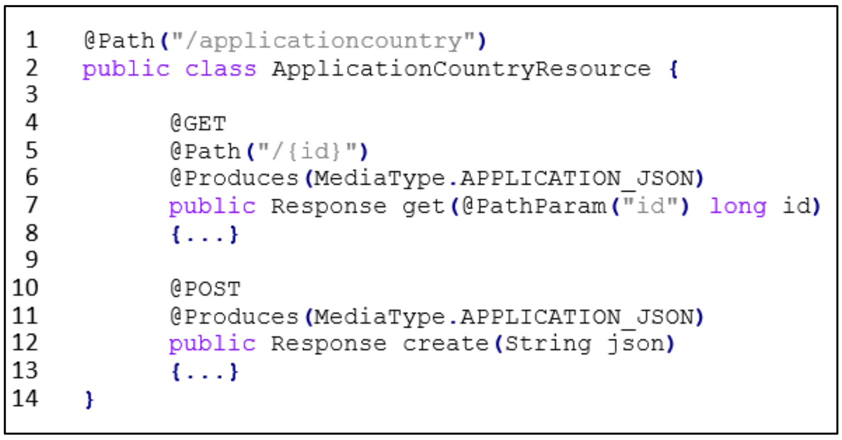
Static contents can be found on the file system on the path specified in the URI, however REST resources are not static content. Instead, they are created within the application server on request. However, before knowing the exact class which should handle the request, the *routing layer* has to determine the target class. This routing is also part of Jersey and is based on annotations. Annotations are *metadata information* added to various places in the source code, and they remain after compilation in the program. Table 9 lists common JAX-RS annotations and how they are used in the solution.

Annotation	Description / Usage

@Path	Specifies the URI-path of a resource. This annotation can be used on the class and on the method level. If used on both in same class they concatenate from class to method level. In our solution the class “ApplicationCountryResource” has the annotation @Path("/applicationcountries")”, and the method “get(long id)” has the annotation @Path("/{id}"). The resulting path for getting a single application country can be seen in Table 7.
@GET, @PUT, @POST, @DELETE	Specifies the HTTP-method used on the request. With the same URI-path multiple methods may be allowed.
@Produces	Specifies the response type. Response types are internet media types [40]. In our software solution we always produce JSON.
@PathParam	Specifies a single parameter from the URI-path. This parameter has to be in the @Path annotation of the method.
@QueryParam	Specifies a single parameter from the HTTP query parameters. Those are placed after the ? sign in the URI.

Table 9: List of JAX-RS annotations used in the software solution

The routing layer of the Jersey framework identifies which class will handle the request based on the annotations. This class will be instantiated (per request) and the method with the matching annotation will be called. If there are parameters (like @PathParam or @QueryParam) the values of those parameters from the URI are extracted and inserted into the method call. After calling the method is responsible for creating a response. However, if the Jersey framework does not find a corresponding method for the request it will respond with the HTTP-code 404 not found. Figure 13 shows a code snippet with annotations of the application country resource class. In line 1 the path for all further methods is set. Lines 4 – 7 show the method to get a single application country resource. Lines 10 – 12 show the method to create a new application country.



```java
@Path("/applicationcountry")
public class ApplicationCountryResource {

        @GET
        @Path("/{id}")
        @Produces(MediaType.APPLICATION_JSON)
        public Response get(@PathParam("id") long id)
        {...}

        @POST
        @Produces(MediaType.APPLICATION_JSON)
        public Response create(String json)
        {...}
}
```

Figure 13: REST annotations on an example resource class

Within a method we typically fetch data from the persistence layer and map it to domain objects. Those domain objects are transformed into JSON, and the JSON is added as string to the response. The HTTP-code is also set for each response according to the result of the call. When we receive JSON data from the client we create domain objects out of the received data, and create or update those entities in the database. Again we create a response. However, in that case we simply either send the identity of the newly created entity as data or no data at all. In both cases we have to set the correct HTTP-code.

4.1.5 Authentication and Authorization

Authentication is the "*…process of verifying someone's identity.*" [41] The data in our application should not be accessible for everyone, only chosen people should access the data. Therefore we have to authenticate each access.

Authorization is the "*process of empowering someone to perform an operation or to have access to restricted resources.*" [41] This is different from the authentication, however, both come together. We need to identify the user who sends the request, and we need to identify what access this user has within our software solution.

We use roles for the authorization of the users. Each user has one or multiple roles assigned. Each role defines the operations available for the user. Table 10 lists the roles of our application and describes the allowed operations for users holding such a role.

Role	Description
Reader	The role specifies read-only access to the data stored in the application. All data can be accessed. However, there are no create, edit nor delete operations allowed. This role can be used for users who just retrieve information from the system.

Editor	The role specifies edit access, which means that all data can be accessed, and inventions (and everything that is modeled based on an invention) can be created, changed or deleted (if delete is available).
Admin	The role specifies edit access as well. However, additional to the possibilities of the Editor-role the maintenance of all reference lists is possible. This is separated from the Editor-role to force the editor to envision changes in reference lists and to discuss these changes with the admin.

Table 10: List of roles in the software solution

The requirements define that the users use the login they already use with other applications within the university network. This is typically called Single Sign On (SSO), where multiple but independent software solutions can use the same access control system, where the user authenticates once against the SSO-system, and with the authenticated session may use other software solutions as well.

The JKU uses Shibboleth as authentication and authorization system. It allows SSO with other web based software solutions, as long as they can provide SAML2 authentication. SAML2 uses profiles to distinguish between communication frameworks and protocols. Although different profiles are available, the JKU uses the Web Browser SSO profile.

In addition to campus wide authentication Shibboleth allows federated authentication, meaning that it is possible to authenticate a user which is not listed in the JKU directory but e.g. in the directory of the Technical University Vienna. However, in the presented solution we do not need the federated authentication.

There are at least two parties involved when doing the authentication via Shibboleth. The *identity provider (IdP)* is the source of the user data, e.g. a directory of all users, and is providing the authentication system. The *service provider (SP)* protects an online resource, e.g. a web based software, and consumes user information from the IdP. Figure 14 demonstrates how the access to a protected resource (Service Provider) is managed:

1. At the beginning the client requests a resource which is hosted at the service provider. The service provider only grants access when the user is authenticated and authorized.
2. The service provider sends the client to the identity provider, where the client has to enter the credentials. On this redirection the service provider adds a generated authentication request.
3. After successful validation of the credentials the identity provider sends the client back to the originally requested resource, although this time the client has the authentication response from the identity provider attached.

4. The service provider verifies the authentication response from the client (originally created by the identity provider) and may grant access to the requested resource.

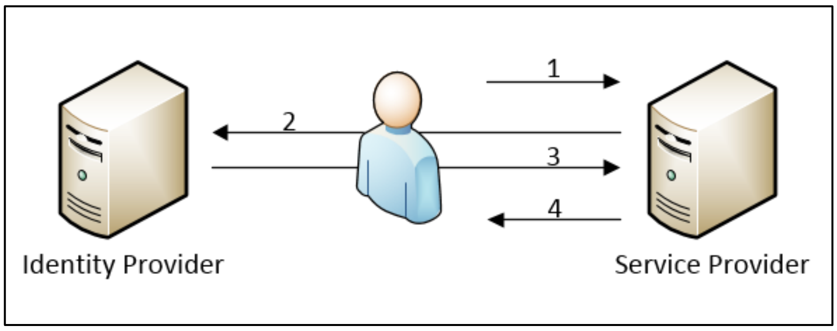


Figure 14: Message-Flow to access a restricted resources

In Tomcat the authentication is done via realms. A realm is a database of users and passwords and their roles. [42] There are different realms available, however, none of the realms is recommended by Shibboleth and therefore they are not used at the JKU.

A common solution is to use the Apache HTTP Server (called Apache) in front of the Tomcat server. All requests will be send to the Apache, which acts as reverse proxy to the Tomcat. Apache is extensible with modules, and there is a module which lets Apache act as service provider for Shibboleth and protects the resources offered.

Our software solution is deployed on the JKU IT infrastructure and uses SSO with Shibboleth with the Apache in front of Tomcat. An overview of the authentication and authorization is depicted in Figure 15, which is in detail described in the following paragraphs.

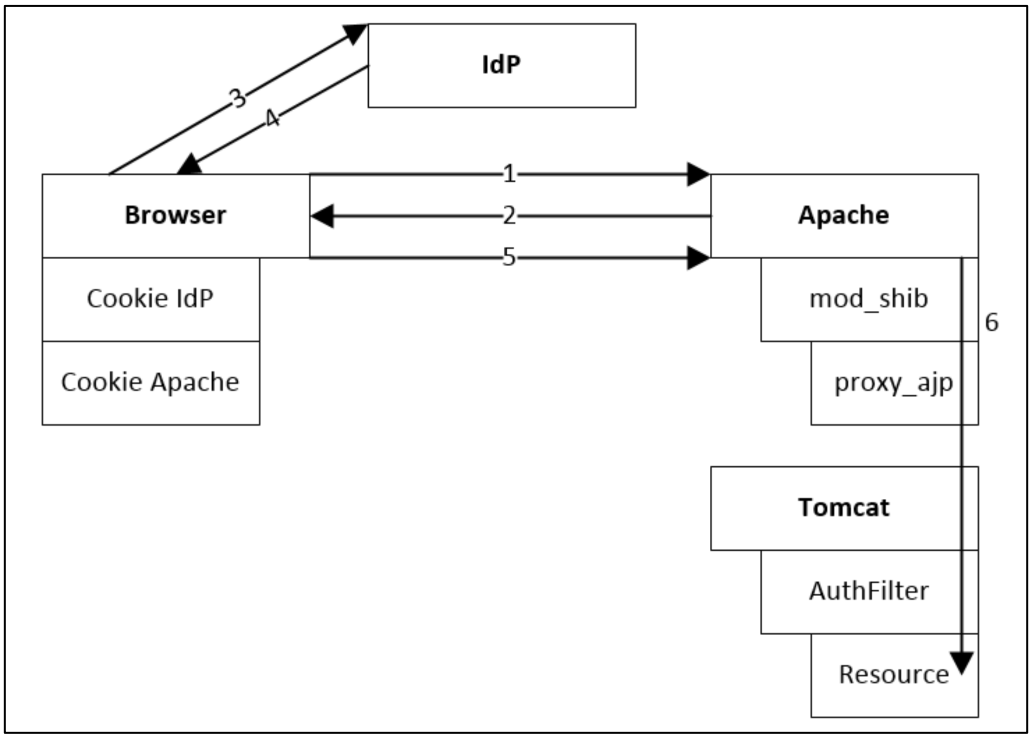


Figure 15: Overview of Authentication with SSO in the solution

1. The request originates from the browser of the client and is sent to the Apache server. Apache verifies whether the client has a cookie from him (within the module “mod_shib”), and if this cookie is still valid. In that case the resource is immediately sent to the client (Step 6).
2. If there was no cookie or the cookie is invalid the Apache sends a HTTP-redirect response (and an authentication request) to the browser. The redirect is the address of the identity provider.
3. The browser sends a request to the identity provider. The identity provider verifies if the client has a cookie from him, and whether this cookie is valid.
4. If the client has a valid cookie, the identity provider generates an authentication response and sends an HTTP-redirect response to the client (with the authentication response). If the client cannot present a valid cookie the authentication server will ask for credentials. After successful authentication the identity provider sends an HTTP-redirect response to the client (with a generated authentication response and with a cookie).
5. The browser requests the resource again. This time the browser presents an authentication response in the request.
6. The Apache module “mod_shib” validates the authentication response. If the validation was successful, the request is passed on to the Tomcat via the module “proxy_ajp”.

Within Tomcat the request is processed with the “AuthFilter”, which generates the security context out of the authentication and authorization information originated from the authentication response. The access to resources in Tomcat (in the Java application) is granted based on roles. The roles the user has are part of the authorization information.

4.1.6 Locking (concurrent editing)

Concurrent editing of the same data will create the issue that the data of the first edit will be overwritten by the second one. To circumvent this problem we implement a custom locking mechanism (see 3.3.3 Multi-platform).

Changing data is only possible if the state of the application allows to do so. There are multiple states which have to be checked:

- Lock on the data: The data may be locked or not locked.
- User of the lock on the data: If the data is locked, the user of the lock has to be compared with the user requesting to edit the data.
- Age of the lock on the data: If the data is locked by another user the age is analyzed.

The data can be edited in this states:

- There is no lock on the data.
- The lock on the data is from the same user.
- The lock on the data is from another user, however, the lock is older than a defined timespan.

The check on the lock is done at the very last moment necessary (e.g. after validation of the received data). The lock is set (or updated) before updating the received data and remains. The REST-API in addition offers an endpoint to request a lock on an entity as well as to remove the lock for each resource. Because we add the check on the age of the lock the data can be edited by a different user even if the lock was not removed in the first place.

When creating new data there will be no concurrent editing, even if the same data is entered by a different user at the same time. However, typically the new client may change the newly created data. Therefore the lock mechanism always locks the created data automatically.

The deletion of data also uses the lock information. In that case the concurrent deletion would be no problem, as one user will receive the successful result and the other one that the entity is not available anymore (which is the same result for the deleting user). The issue may arise if one user edits the data and during that process another user deletes the data. When the first user saves the data the entity is not available anymore and the user will receive an error message. Therefore delete methods also require the successful setting of a lock. After deletion of the entity the lock can be removed automatically. Even if a user requests the data for editing exactly during the deletion process, the user will receive an error message that the data is not available for editing.

The lock information has to be stored during the lifetime of a lock. This information could be stored within the memory of the application server. However, this memory information will be lost during the restart of the web application. Although this restart should not happen during normal operations, the possibility still exists. Therefore we decided to store the lock information also in the DBMS.

The persistence layer automatically creates information about modification (user, date and time) on each entity. If we store the lock on the individual entity we will update those data, even if there was no edit at all. This will therefore change data and render this data useless. Another solution is to use a separate table to store the lock information. With this table in place the information of the entities will not be changed by lock mechanism. We implemented the custom locking solution with a separate table as persistent storage and a configurable timespan to free locks when they are not removed by the client.

4.1.7 Static content

Static content are file-resources, which are stored on the web server and not dynamically generated. If such a resource is requested it is sent to the client just as it is. Because they may only change with new releases of the software solution those files can be cached on the client, which may increase the loading time of the page, and decrease necessary server resources. The software solution uses different types of static content, which are described in Table 11.

Annotation	Description / Usage
HTML-Pages	This are full pages, which do not change during the life cycle of requests / responses.
CSS	These files describe styles which are used in the web application.
JavaScript	These files contain code which may be executed on the client side. The files are either implemented by third parties or by us. If they are implemented by us, they are application specific. However, they may contain code which can also be used on other applications. If they are from third-parties, then they can be used on other applications as well and are

	used as additional components for our software solution.
Images, Icons	For the visual appearance sometimes icons or other images are used.
HTML fragments	The SPA loads after initial loading of a full HTML-page only HTML-fragments from the server. Those fragments are shown as Views on the client.

Table 11: List of types of static content in the software solution

Our software solution with the SPA on the client and REST-service on the server side even allows to place static content on another web server than the one where we install the REST-service. In our case this is not necessary.

4.1.8 Logging

The purpose of logging in the application is to have information about events that occurred and data about past actions in the application. The logging of the application is based on log4j, which is a library of a logging framework.

During the lifecycle of a request the application sends information about the request itself, the incoming payload (JSON), the methods executed, the user issuing the request, and various other information to the logging framework. The logging framework allows the persistence of the information in files, which can be analyzed if there are unexpected behaviors. The authentication filter logs also the duration the server took to process the request, because the authentication filter processes the request and the response within the same instance. During the initialization of the request, on the server-side, a unique identifier is created, which can be used to track the individual log messages of a single request. The details about the created log files can be found in the Appendix.

4.1.9 Server-side libraries

Today the developer can access a plethora of components developed by other people and use that components in software solutions. The advantage is, that this lowers the development time, and greatly reduces the error-rate of a software solution. A component may be used by a lot of other developers as well, and this increases the discovery of bugs and therefore, when they are fixed, the quality of the component. Some components also use other components, meaning, that if the developer want to use component A, which uses component B, he has to include both components, even if he does not use component B directly.

In our software solution we use also components from other developers or organizations. We always have to check the license or the rights associated with using the component. Table 12 shows the various components used on the server side of our application. The version which is used and the license the component uses are also cited.

Component	Version	Description / License
HK2	2.4.0-b34	Dynamic dependency injection framework *License: COMMON DEVELOPMENT AND DISTRIBUTION LICENSE (CDDL - Version 1.1) and GNU General Public License (GPL - Version 2, June 1991) with the [“Classpath Exception”]*
ASM	5.0.4	Java bytecode manipulation and analysis framework *License: individual license* [43]
Jackson	2.7.3	JSON library for Java *License: ASL, Version 2.0*
Javassist	3.18.1	Java bytecode manipulation *License: MOZILLA PUBLIC LICENSE*
Javax	Various versions	Additional Packages for java from Oracle *License: COMMON DEVELOPMENT AND DISTRIBUTION LICENSE (CDDL - Version 1.0)*
JAXB-API	2.2.7	Reference implementation of JAXB *License: COMMON DEVELOPMENT AND DISTRIBUTION LICENSE (CDDL - Version 1.1) and GNU General Public License (GPL - Version 2)*
Jersey	2.22.2	Framework for RESTful web services in Java. *License: COMMON DEVELOPMENT AND DISTRIBUTION LICENSE (CDDL - Version 1.1) and GNU General Public License (GPL - Version 2, June 1991) with the [“Classpath Exception”]*
jOOQ	3.7.3	ORM Framework *License: ASL, Version 2.0*
Log4j	2.5	Logging Framework

		License: ASL, Version 2.0
PostgreSql-JDBC	9.3.1102	JDBC driver for accessing a PostgreSQL database. *License: BSD License*
Validation-API	1.1.0	Bean validation API *License: Apache 2.0*

Table 12: List of external components used on the application server

Although not all libraries are used directly within our code they may be required and used by libraries we use.

4.2 Database

"A large amount of data stored in a computer is called a database. The basic software that supports the management of this data is called a database management system (dbms)." [44]

In our software solution we have to store the data, and we store this data in a database. As already described in the previous section we use an ORM-tool to access the database and retrieve or change data on that database.

4.2.1 Relation between Domain Model and Relational Model

Before we can use the database it has to be created. We decided to use a relational DBMS, therefore we have to create a relational model of our data. This model is similar to our domain model. Based on the requirements we created our domain objects and the properties. In the relational model we have to create tables and columns, and model the relations between them.

As already described in 4.1.2 Domain Model there is often a one to one relationship between a domain object and a database table. However, in object oriented programming often inheritance is used, and in addition a class can have multiple different usages. In our domain model we have e.g. the class "ChooseItem", which is used for holding information about dropdown-elements in the user interface. Those values come from various reference lists (e.g. the list of special fields the university has). If we want to store this information in the database, we have two possible solutions: one table for each reference list, or one table for all where we have to use a discriminator column. A discriminator column is used to store data of different type within the same table.

Creating only one table to hold different types of dropdown values looks upfront nice, because it is initially less work. However, in the future there may arise the need to add more columns for a specific type. And then the need to add another columns for the other type. This will make a table with a lot of columns, however, they are not used for every row. Figure 16 shows such a table on top of the figure, named "commondropdownvalues".

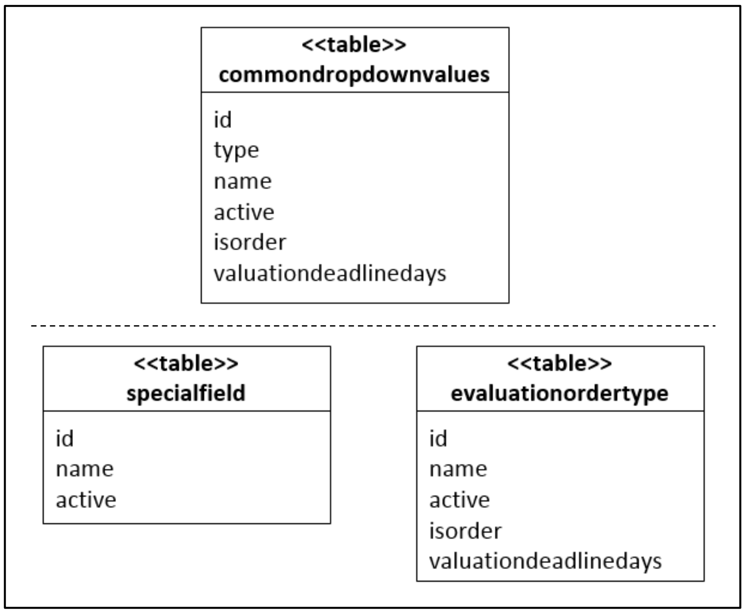


Figure 16: Possible solutions for similar tables

On the other side, if the different types are known upfront we can create a table for each type, even if they have the same columns (see Figure 16 where specialfield and evaluationordertype are such tables). When new columns have to be added we do not have to bother with the remaining data of different type, because we store only one type of data in the table. We can easily modify the schema of the table based on our requirements. During our project we presented various stages of the software solution, and because the end users could access these stages directly they already ordered changes. These changes were easily included in our relational model, because we designed it upfront with one table per business domain object, not using tables for more than one type of data.

4.2.2 Deleting data

In the requirements there was also the deletion of data stated, although in discussion with the end users they also wanted a fallback solution if they delete something inadvertently. This means that the software solution should not delete data at all. However, when the data is not physically deleted from the database it has to be marked somehow.

There are various solutions for this issue. One common solution is to move the data to another table. With that solution only none deleted records are in one table. Another solution is to use a Bit-column (or Boolean) to distinct between deleted and active records. Restoring a deleted record is done easily by just changing the deleted-mark, whereas in the two table solution data has to be moved. When using automatically incremented primary keys (like we do in our solution) this will bring up various additional tasks (e.g. foreign-key changes). We decided to use the delete-mark on the tables where we store entities which may be deleted by the end user.

4.3 Angular Application

4.3.1 Overview

There are various concepts used in an Angular application. The architecture overview of an Angular application is shown in Figure 17.

An application can consist of multiple modules. Each module has a configuration, routes are used to identify the view and controller to use. Views are templates, based on HTML with additional markup, and views are rendered in the browser. They use directives, which extend HTML with attributes and elements. The controllers add business logic to the views, and the view and the controller have a shared scope, where the view and the controller can access a model. Controllers may use factories and services, which are reusable components containing business logic. However, they are independent of controllers. [45]

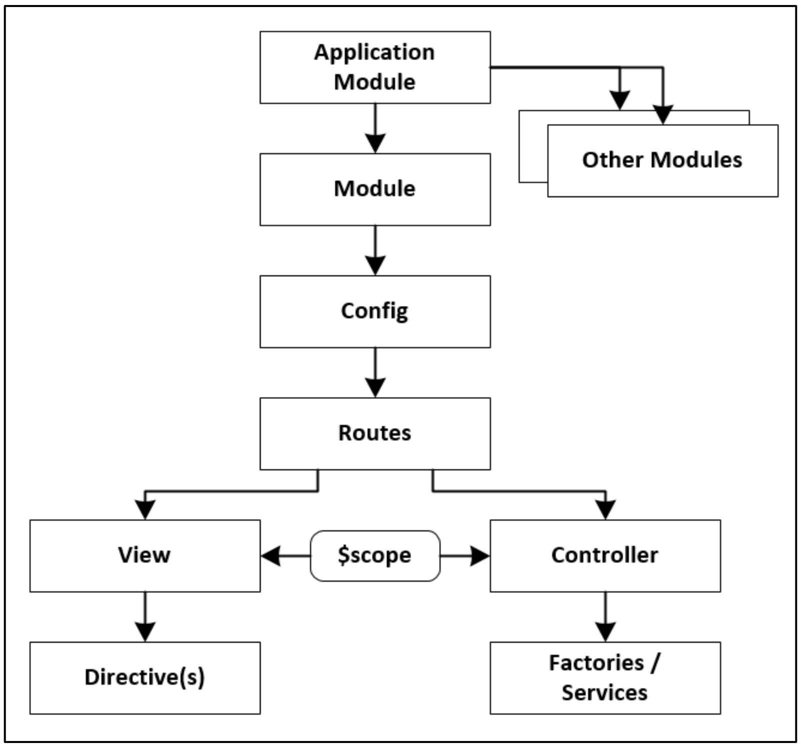
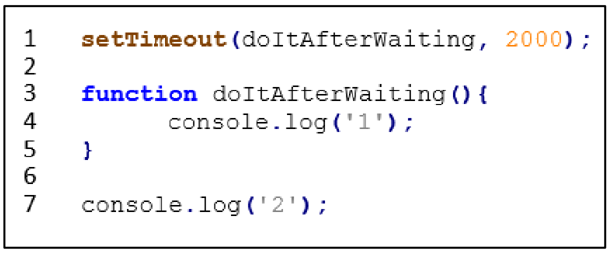


Figure 17: Architecture overview of an Angular application [46]

One of the main concepts of Angular (not shown in the Figure) is *dependency injection*. “*Dependency injection is the concept of asking for dependencies of a particular controller or service instead of instantiating them inline via the new operator or calling a function explicitly…*”. [47] An example of dependency injection in Angular can be seen in Figure 22 (lines 2 – 5).

Another concept, which is basically a JavaScript concept, is *asynchronous operation*. Some methods use callbacks. This means, that the method is called and afterwards the caller has no direct feedback or result from that method call. However, if he provides on the call of the method a callback (which is in fact another method, even with parameters), the called method will call the callback. As example in Figure 18 the method “setTimeout” is called. This method expects a callback and the time to wait until the callback is called. Running this example will first output “2” on the console, and after 2 seconds “1” will be printed out.



```
setTimeout(doItAfterWaiting, 2000);

function doItAfterWaiting(){
	console.log('1');
}

console.log('2');
```

Figure 18: JavaScript example of an asynchronous callback

In the following sections the concepts of Angular are described in more detail, and it is also shown how these concepts are used in the software solution. In addition the client side of the authorization is presented.

4.3.2 MVC and MVVM pattern

One of shortest definition of the MVC pattern can be found in Fowler’s book:

“Splits user interface interaction into three distinct roles”. [23]

The *model* is an object that represents and maintains data. This object is not displayed in any form. An example is the application country, which was received from the server. The *view* is the part that is displayed in the user interface. An example is the HTML which is used as template for the browser to render. The *controller* maintains the connection between the view and the model and handles the interaction, e.g. change of the model through an input of the user. For example after clicking on a button in the user interface the controller handles the click and sends the data to the server. The controller in combination with the view can be seen as the user interface. The structure of the pattern is shown in Figure 19. [23]

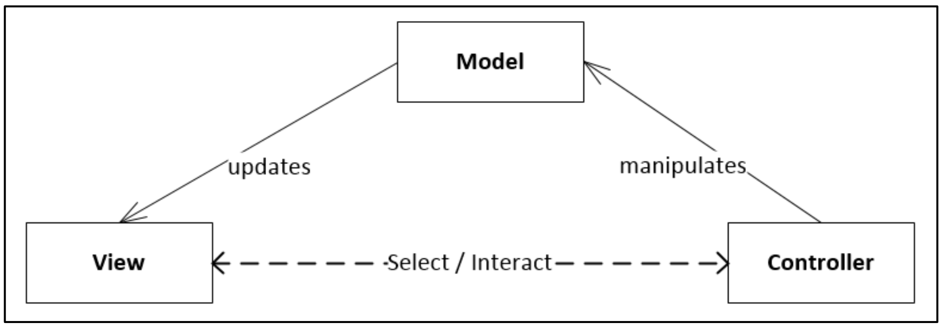


Figure 19: Model-View-Controller (MVC) pattern

The biggest advantage of the MVC pattern is the separation of concerns, meaning that the individual roles have distinct concerns. We separated the presentation from the model therefore we can easily test our model, whereas user interface testing is more complex. Often the user wants to see different representations of the data. This can be achieved by using the same model with different views. The most important rule is that the presentation depends on the model, however, the model does not depend on the presentation. This separation has to be implemented, otherwise the positive effects cannot be gained. Fowler argues that the separation of the view and the controller is not that important. [23]

As MVC is a pattern, there are also variations of it, because it has to be implemented in the environment the developer uses. When using Angular the MVC pattern is used, however, in a slightly different way: MVVM – Model-View-ViewModel. The exact flavor cannot be defined, because the differences are often only very small or philosophically.

In the MVVM pattern the roles are still distinctive. The view and the model have the same role as in the MVC pattern. An abstraction of the view is created with public properties and commands. This abstraction is the view model. To stick the view model with the view together databinding is used. Because Fowler states that the distinction between view and controller is not very important, often view and controller are very close together. In MVVM the view has to be seen as view and controller. However, the view model can also be seen as specialized controller which changes model information into view information and passing commands from the view to the model. The presentation logic, which may be part of the controller, is separated into the view model. [48] [49] In Figure 20 the structure of the pattern is shown.

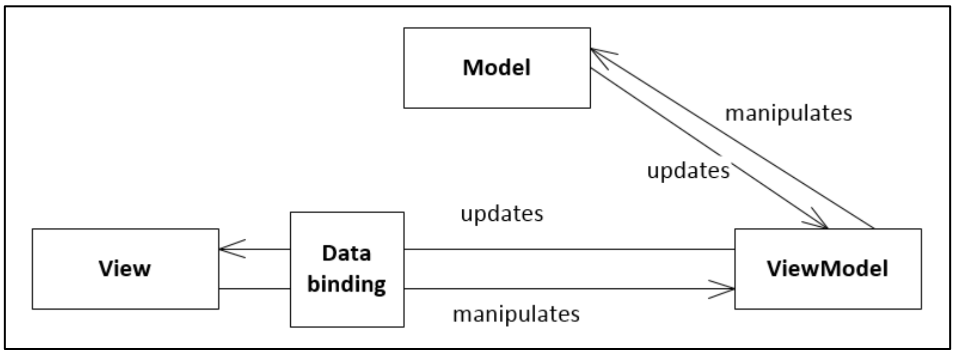


Figure 20: Model-View-ViewModel (MVVM) pattern

One of the popular features of Angular is the two-way databinding. This means, that the view model is automatically updated when the value in the view is changed, and that the view is automatically updated when the model changes. The code snippet in Figure 21 shows an example of two-way databinding. The properties “qty” and “cost” are declared in the initialization-block (line 1), which acts as the view model. The two-way binding is established with the attribute “ng-model” (lines 4 and 7) which is added to the HTML attributes of an Angular app. This attribute defines the property on the model for the individual input element. Because two-way databinding happens immediately the view model properties can be used without further interaction like pressing a button.

```
1   <div ng-app ng-init="qty=1;cost=2">
2     <b>Invoice:</b>
3     <div>
4       Quantity: <input type="number" min="0" ng-model="qty">
5     </div>
6     <div>
7       Costs: <input type="number" min="0" ng-model="cost">
8     </div>
9     <div>
10      <b>Total:</b> {{qty * cost}}
11    </div>
12  </div>
```

Figure 21: Angular two-way databinding [45]

In the Angular application the views are HTML-templates. They contain the markup for the user interface. This markup is enhanced with the Angular specific attributes (like ng-model), databinding elements (interpolation) and directives (see 4.3.4 Modules, Services, Filter and Directives). During the life cycle of the Angular application and especially the view the different elements are observed and if they have expressions assigned those expressions get evaluated and the results are presented in the user interface.

The interpolation construct can be found in line 10 of the code snippet in in Figure 21. The curly brackets are used for databinding. The expression within the curly braces is evaluated and may contain just a reference to a view model property or a valid JavaScript expression, which also has access to the view model properties. In addition to this the expression may also contain additional information, e.g. for formatting the result. This is done with filters, which are JavaScript methods themselves (see 4.3.4 Modules, Services, Filter and Directives).

As often seen on previous web applications where JavaScript and HTML are used next to each other and HTML-pages typically contained JavaScript areas, this is not the case in Angular. The views only contain references to the view model. In Angular the connection between the view and the view model is referred to as the scope. Components (like controllers and views) have their own scope, however, if nested it is possible to create shared scopes.

In Angular controllers are used as the container for view models. Controllers are implemented with JavaScript. The main responsibilities of the controller in an Angular application are: [47]

- Getting the necessary data from a data store for the view,
- evaluating what parts of the data should be shown,
- presentation logic, and
- handling user interactions, like a click on a button.

As presented in the next section, we typically inject the necessary data on loading the controller. However, due to user interactions it may become necessary, that the controller retrieves additional data, but the task of getting the data is done in a service and the controller just uses the service (e.g. the “applicationCountryService”).

In Figure 22 a code snippet of one controller is shown. The responsibility of this specific controller is the user interaction and view of editing a single application country. Every controller is named (line 1). This name is used from Angular to reference exactly that controller. Lines 2 – 5 contain information for the dependency injection (see 4.3.1 Overview). The required components are specified as text (lines 2 - 3) and method-parameters (4 - 5). The reason for the two time declaration is the minification support. Minification is the process of shrinking the size of JavaScript or HTML files by removing all unnecessary information (e.g. comments, line breaks etc.), and in JavaScript also by renaming properties and method parameters to short names. In the case of the presented controller snippet the result of minification may look like shown in Figure 23. The minified code will still be valid, however Angular will not be able to inject the correct dependencies, hence it will not work as expected. Dependency Injection depends on type of the needed dependency, nut JavaScript does not use types to define properties or method parameters. Without type definition the dependency cannot be resolved. Therefore when declaring a method that will receive injected components Angular allows this definition before the method with strings, because strings will not be changed during minification.

In line 7 we define our view model. Because the controller is also used as view model, the reference is on "this". This syntax allows the definition of members which should be public to the view. Those members can be used for data bindings. This syntax (using "vm") has three advantages [50]:

- A consistent and readable method of creating view models in the controllers
- Issues with using "this" are gone (issues could arise e.g. when using closures within controllers)
- $scope is not used anymore. This helps when nesting views and controllers.

The property "applicationCountry" is defined in line 8 and will be assigned in lines 11 - 13. This assignment will only be done if there was an "id" in the state parameters (line 10). The state provider (see 4.3.3 Application state and routing) will inject the parameters into the method. The list of application countries is also injected into the method. If the application country with the given "id" cannot be found an error message will be displayed (lines 14 - 17). This is done with an "alertService", which is again injected and will be described in a following section.

If there was no "id" injected, the controller will use a new application country object. The new object is created in the "applicationCountryService" (which was also injected) (lines 18 - 20). The "submit" method may be called from the view and contains the implementation of handling the save operation (omitted in this snippet) (lines 22 – 27).

```
angular.module('invAdminApp').controller('editApplicationCountryController',
['$scope', 'baseData','$filter','$stateParams',
'applicationCountryService','alertService',
function ($scope, baseData, $filter, $stateParams,
applicationCountryService, alertService) {

      var vm = this;
      vm.applicationCountry = {};

      if ($stateParams.id) {
            vm.applicationCountry =
                  $filter('getById')
                        (baseData.applicationCountries, $stateParams.id);
            if (!vm.applicationCountry) {
                  alertService.addError(
                  'Das gewählte ...');
            }
      } else {
            vm.applicationCountry = applicationCountryService.createNew();
      }

      vm.submit = function(form) {
            alertService.clear();

            applicationCountryService.saveData(vm.applicationCountry)
                  ...;
      };
}]);
```

Figure 22: Example of a controller for editing an application country

```
function (a, b, c, d, e, f)
```

Figure 23: Minificated method declaration

4.3.3 Application state and routing

In the application there are multiple views and controllers, yet the application has to decide when to show what view. In Angular this is done via application state and routing. In the starting phase of the application the routing is configured. For every application state the view and the controller among other settings are defined. The URI for the application state is also defined and allows calling a specific application state through browsing to the URI. A single state can be as simple as shown in Figure 24 in lines 2 to 4. The view is defined with the “templateUrl”, and the URI-path of the state is defined with the “url” parameter. Angular caches views on the client. On the first usage of the view the template is loaded from the server, afterwards it is used from the cache, which decreases load times (no server interaction necessary).

The second state can be seen as an example of our software solution. The specified view is responsible for managing a single application country. Because we have user interaction we need the controller, where we define the name of the controller (line 9). In the view we need to access our view model, which is part of the controller. Therefore we want to have a name for the controller used in the view (line 10). If we do not specify the name, we may get in trouble with mixed scopes.

We may load all controllers on startup, although it is better to load – like Angular does automatically with the view – the necessary controller only when needed. This can be done with the component “ocLazyLoad”, which allows lazy loading of static content from the server (lines 13-22). The view will present a single application country. The data of the application countries is stored in the database, and may be accessed by the REST-API on the application server. The loading of the data is implemented in the “applicationCountryService” within the “loadData” method.

The controller may load the data himself. In that case we have the loading code in the controller, and if we have to load the application countries in another controller as well, we have the code on two places. This can be avoided by using a service (see 4.3.4 Modules, Services, Filter and Directives). Still, with the service implanting the loading of the data, the controller can now use the service to load the data. The application flow will be loading and displaying the view, controller using the service to load the data, when data arrives the view will be updated. This will lead to a view displayed to the user without actual data, because the data is still be loaded. To circumvent this issue it is possible to load the data before loading and displaying the view (lines 23 – 26). Actually, the state change only occurs after the data is loaded. If loading of the data fails, the state will not change.

```
1   $stateProvider
2           .state('dashboard.grid',{
3                   templateUrl:'views/ui-elements/grid.html',
4                   url:'/grid'
5           })
6           .state('dashboard.editApplicationCountry',{
7                   templateUrl:'feature/applicationCountry/applicationCountry.html',
8                   url:'/editApplicationCountry/:id',
9                   controller:'editApplicationCountryController',
10                  controllerAs: 'ctrl',
11                  resolve: {
12                          deps: [
13                                  '$ocLazyLoad', function ($ocLazyLoad) {
14                                          return $ocLazyLoad.load([
15                                                  {
16                                                          name: 'invAdminApp',
17                                                          files: ['controller.js']
18                                                  }
19                                          ]);
20                                  }
21
22                          ],
23                          baseData: ['loadMyServices','applicationCountryService',
24                            function (loadMyServices, applicationCountryService) {
25                                  return applicationCountryService.loadData();
26                          }]
27                  }
28          })
29  ;
```

Figure 24: Angular state and route configuration

Angular offers multiple ways to change the state. The most common one is using the URI. The router will evaluate the URI and looks for a matching pattern in the configuration. When a state is found this state will be enabled (if all dependencies can be found and evaluated), otherwise a state change error occurs. There is the possibility to create links within the application. This link contains the URI of the target, within or outside of the application. In the first case Angular offers a directive (see 4.3.4 Modules, Services, Filter and Directives) which allows the specification of the application state instead of the URI. Named parameters could also be added. Figure 25 presents an example of the Angular application, where the state “editInventionInvoice” will be called upon using the link. The parameters for the state are given in line 2. In line 3 is the content of the link, which is an icon of the Font Awesome package [51].

```
1   <a ui-sref="dashboard.editInventionInvoice(
2                  { inventionId: ctrl.invention.id, id: invoice.id }
3           )"><i class="fa fa-pencil"></i></a>
```

Figure 25: Linking to the application state

The state provider (which is the routing mechanism in our application) also emits events. Those events can be catched when configured. The most common event to catch is the “$stateChangeError”. If this event is emitted it is possible to show an error message on the user interface notifying the user of the error happened.

4.3.4 Modules, Services, Filter and Directives

4.3.4.1 Modules

The concept of modules is the way Angular packages relevant code under a single name. In Java this is done with packages. As already described in the overview modules may depend on other modules and contain controllers, services, factories, filters and directives. [47]

4.3.4.2 Services

As already mentioned above, Angular applications use services. Angular defines a *service* as a *reusable component with business logic* and independence from the views. Although most of the operations can also be done within the controllers the separation into reusable components will reduce code duplication. Although the term “service” is used as “reusable component” Angular allows the creation of factories, services and providers, each of them having a different meaning in the Angular application. [47]

In our Angular application the services are responsible for loading the data from the server, prepare the data to be usable in the controller as view model, and finally for sending data to the server if there were changes in the data. The operation to change data on the server has to be explicitly initialized from the user (e.g. by clicking on a button). Figure 26 shows a code snippet of a contract (public interface) of one of our services. Our services are centered on features, where a feature is the combination of listening and editing a single domain object, e.g. the application country. Notable are the methods in lines 2 – 5, the others were already described in a previous section.

```
function createNew()
function loadData()
function extractData(data)
function enrichData(data)
function saveData(data)
```

Figure 26: Example contract of a service in our software solution

The “loadData” method is responsible for loading the necessary data from the server. This may be just a single call to an REST-API endpoint, or the call to multiple endpoints. Nevertheless, when receiving the results from the server the data has to be converted to a JSON object, and individual domain objects have to be extracted. This is done in the method “extractData”. Sometimes we have to enrich our data (done with method “enrichData”). It means we have to add additional data based on the data received, e.g. when showing invoices we also want to have the sum of all invoices. If done on the server we need an additional property on the list, which is generic, meaning we need an inherited list as well. On the other side, when the client needs the sum it could also been calculated easily there. This should be reusable, independent of the controllers using the data. Therefore the service performs this task. The “saveData” method does not have any user context on the client side. The server extracts the user information from the authenticated request and uses that information for the “persist(userId)” method.

An important fact is the lifetime of services. A controller is created when the view is shown, and on changing to another view the controller is destroyed. A service is only created once, which means it is a *singleton instance*. [52]

Services are not only used in the way that they accompany controllers as reusable components. Because they are singletons they can also be used for components which should have a persistent state between view (and therefore controller) changes. Our application uses therefore an alert service, which has internally a list of messages (either error or success), which maintains its state through state changes. However, this is not the only advantage. Because it is a singleton, the same instance is injected throughout our application. E.g. when calling a service from the controller, the service can also use the alert service to add error messages, even if it has no idea about the view displaying that message or the controller calling him.

4.3.4.3 Filters

Angular also contains the concept of Filters. “*AngularJS filters are used to process data and format values to present to the user.*” [47] Angular contains various filters to format e.g. data to numbers or dates. The filters can be used on expressions in our templates and also in the controllers or services. They can be inserted like any other service with dependency injection.

In Figure 27 two filter usages are shown. The first one (line 1) is a predefined filter from Angular to format date. Filters can also have parameters. The second example (line 2) is a filter included in our software solution, which will shorten the value to 50 characters and add “…” if the value has more characters. The view-model is not changed, the filter is just applied before displaying the data without changing the data itself.

```
<td>{{ invoice.entryInvoiceOn | date : 'dd.MM.yyyy'}}</td>
<td>{{ invoice.contractor | shortDesc }}</td>
```

Figure 27: Example of Angular filter usage

The usage of a filter within a controller can be seen in Figure 22 on lines 12 and 13. This filter is also one of our included ones. All of our domain objects have an identity property, the “id”. This property is also transformed to JSON, therefore all JSON objects have an identity property. The filter is used to extract one entity out of a list of entities by comparing the idenitity property with the given id.

4.3.4.4 Directives

Angular defines the concept of directive as an extension to HTML with custom attributes and elements. That means that either it is possible to add new HTML-elements (HTML-tags) or to add attributes to existing HTML-elements.

As seen in Figure 21 there is the attribute “ng-model” which is not an attribute of the HTML-specification. During the life cycle of an Angular view those new elements and attributes are replaced with the result of the evaluation of their implementation. Angular has already various directives included which build the base for the implementation of an application. However, it is possible to create own directives for reusable HTML-elements or for adding functionality to existing HTML-elements.

Figure 28 shows a HTML-fragment where one of the directives we added to our Angular application was used. In the domain model we often have Boolean properties, most of them holding one of three values (empty, true or false). In the user interface we do not want to display the concrete value (which is empty, true or false), nor do we want to show the values “yes” and “no”. Dependent on the displayed data and the property we may have different meanings for the values. Instead of always writing a JavaScript expression to evaluate all states (code duplication as well as JavaScript in the HTML-fragments) we added a custom directive. This directive is loaded on application start and available for all views in the application. This way we have a declarative component. Another common directive of our application is the list count directive, which adds the information about the number of elements in lists to the user interface.

```
<nice-true  nice-true-text="liegt auf"
                  nice-false-text="nicht vorhanden"
                  nice-null-text="nicht vorhanden"
concrete-value="ctrl.invention.inventionNotificationPresent">
</nice-true>
```

Figure 28: Example of Angular custom directive usage

4.3.5 Authorization

Although the authorization is checked on the server side when receiving requests the client also needs the authorization information and a permission management. This is necessary to show the user only those elements which he is allowed to use. Usually the user interface contains e.g. a link to an edit view, except when the user has only the role “reader”. When clicking the link we have to check the permission and display the message “you are not allowed to use this view”. Nevertheless we have to implement this check, but it would be better to show only possible operations within the user interface and hide those operations that are not allowed for the current user.

The library “angular-permission” offers the functionality necessary for our application. The Angular-application requests the information about the currently logged in user on startup from the server. The server finds this information in the authentication and authorization information from the SSO service. The roles (see 4.1.5 Authentication and Authorization) and the respective permissions are also defined on the client side and assigned during the startup phase of the client application.

Basically there are two issues which are addressed. On the one side there is the check that a view or operation is only accessed by authorized users, on the other side we want to hide operations (e.g. editing an invention) which are not allowed for the user.

The library works in conjunction with the router library. It is possible to put the permission information for a view directly in the route definition. When the state change happens, which is the case when the user clicks a link or opens an URI, the permission is read from the route definition and checked against the permissions the current user incorporates. If the permission is not available for the current user another view is presented instead of the requested one. This particular view will respond to the user that he does not have enough privileges.

The library offers, in addition to the router-integration, also directives to use in the markup of the views, thus it is possible to restrict operations or even areas of the markup to those users who have the permission. The library itself uses the “show” and “hide” CSS-classes which are also used from Angular-directives, thus we have to introduce sometimes an additional HTML-element to surround the element which we want to protect with permissions.

4.3.6 Locking (concurrent editing)

In section 4.1.6 the implemented custom locking mechanism to avoid the concurrent editing of the same entity is shown. As described the client has to request a lock for editing, or may send the edited data and receive the result of the operation, which may fail if the server cannot acquire a lock.

The Angular application shows the data in read-only views, and offers a button to change the data, which will, if pressed, present the edit view. During this state change the application requests a lock on the requested data from the server. Only if the lock can be acquired the edit view will be presented.

Locks are retained until they are removed or overwritten by other locks. During the lock-period no other user can edit the locked data. This means that the user has to have the ability to remove locks. Because Angular controllers are instantiated every time they are used and Angular provides life-cycle events we decided to use the destructor event ("$destroy") of the controllers, to remove the lock of the previous shown entity. This event is almost always fired. However, if the user closes the browser (or just the browser tab running the application) the event will not be fired. In that case the entity will be locked until the lock mechanism acquires another lock. If the same user again requests the edit of the entity (e.g. after a client-computer crash) the request will be successful.

4.3.7 Client-side libraries

As already described in a previous section modern software development also favors using components if applicable instead of developing everything from ground up new. The Angular module system allows also the import of modules from other developers, and JavaScript itself is also very popular and there are various components based on JavaScript which may be added to an application. Even Angular itself is modular and not all modules have to be used (and loaded).

We always have to check the license or the rights associated with using external components. Table 13 shows the various components used on the client side of our application. The version which is used and the license the component uses are also cited.

Component	Version	Description / License
angucomplete-alt	2.4.1	Autocomplete directive, input box used in assigning persons as inventors to inventions. *License: MIT license*
angular	1.5.6	The base angular framework. *License: MIT*
Angular-animate	1.5.6	Adds animation support.

		License: MIT
Angular-base64	2.0.5	Adds Base64 encoding and decoding support. *License: custom, provided as is and can be used, modified etc. as long as original license is included*
Angular-bootstrap	1.3.3	The bootstrap components written in AngularJS *License: MIT*
Angular-confirm-modal	1.2.5	Adds a confirmation dialog for Angular. *License: ASL 2.0*
Angular-cookies	1.5.6	Adds cookie support. *License: MIT*
Angular-datatables	0.5.4	Angular wrapper for datatables.net *License: MIT*
Angular-loading-bar	0.9.0	Adds a loading bar to the user interface when asynchronous HTTP requests are send. *License: MIT*
Angular-messages	1.5.6	Adds support for displaying validation messages in the UI. *License: MIT*
Angular-mocks	1.5.6	Adds support to inject and mock Angular services into unit tests. *License: MIT*
Angular-permission	3.2.1	Adds role and permission support to ui-router and an Angular Application. *License: MIT*
Angular-resource	1.5.6	Adds support for using RESTful services. *License: MIT*
Angular-sanitize	1.5.6	Adds support to sanitize HTML

		License: MIT
Angular-touch	1.5.6	Adds touch events for touch-enabled devices. *License: MIT*
Angular-ui-grid	3.2.1	Adds a data grid for Angular. *License: MIT*
Angular-ui-router	0.3.1	Adds routing support to Angular (instead of the default Angular router) *License: MIT*
Bootstrap	3.3.6	HTML, CSS and JS framework for web projects. *License: MIT*
Datatables.net	1.10.12	Adds a data grid (for HTML and JavaScript) *License: MIT*
Datatables.net-responsive	2.1.0	Adds responsive extension support to Datatables.net *License: MIT*
Es5-shim	4.5.8	Adds EcmaScript support for older browsers. *License: MIT*
Font-awesome	4.6.3	Adds fonts and icons to a web application. *Licenses: SIL OFL 1.1, MIT, CC BY 3.0*
Jquery	2.2.4	JavaScript library for DOM manipulation, event handling, animation, etc. *License: custom, provided as is and can be used, modified etc. as long as original license is included*
Json3	3.3.2	Adds a JSON implementation. *License: custom, provided as is and can be used, modified etc. as long as original license is included*

metisMenu	2.5.2	A collapsible vertical menu JavaScript *License: MIT*
Oclazyload	1.0.9	Adds lazy load support to Angular. *License: MIT*

Table 13: List of external components used on the client side

The dependencies are not manually managed. We use the package manager "bower.io", which uses a JSON-file for declaring dependencies. Afterwards all dependencies are loaded from their individual repositories and placed within a folder of the web application.

4.4 Program Flow from User-Interaction to UI-Feedback

In this section we want to point out an example of the flow of the software solution from handling a user-interaction (like the request to get the list of application countries) to the result (updated user-interface). Most of the parts involved were already described in previous sections. The following description is split into the client and the server side.

4.4.1 Client side

The user has to use the browser to access the web application. To use the application, the user has to authenticate with his credentials on the Shibboleth-page, which is provided by the university. After the successful authentication the user can issue requests within the application. An overview of the program flow on the client-side is shown in Figure 29.

The user has to click on a menu entry in the presented HTML page in his browser. This will trigger the URL-router to change the state of the application (see 4.3.3 Application state and routing). Therefore the router has to determine the view and controller which fit to the requested menu entry. The request for data from the server will always be asynchronous. This means, that the view does not have the data when initially presented to the user. To circumvent the empty view issue the URL-router allows the loading of the data upfront. The router will wait until the data, concerning the client, is available to present the view to the user.

The loading and preparation of server side data is always initiated from services (see 4.3.4.2 Services). Those components can be reused on multiple controllers or even on other services. The services use the Angular service "$http" to request data from the server. As the services often need more than one data from the server, multiple requests are combined. This can be done with the use of "$q", where the result is available after all requests are fulfilled (promise resolved). The data is retrieved from the REST-API, where the different data has different URI.

After successfully loading the data the URL-router instantiates and loads the view and the defined controller. The URL-router injects the loaded data among other services into the instantiated controller. The URL-router has to load the controller JavaScript-file and the view-template from the server if it is not already loaded on the client.

In the controller the ViewModel is created and the data is associated to the ViewModel (see 4.3.2 MVC and MVVM pattern). The view can access the ViewModel and provides the visual result to the user.

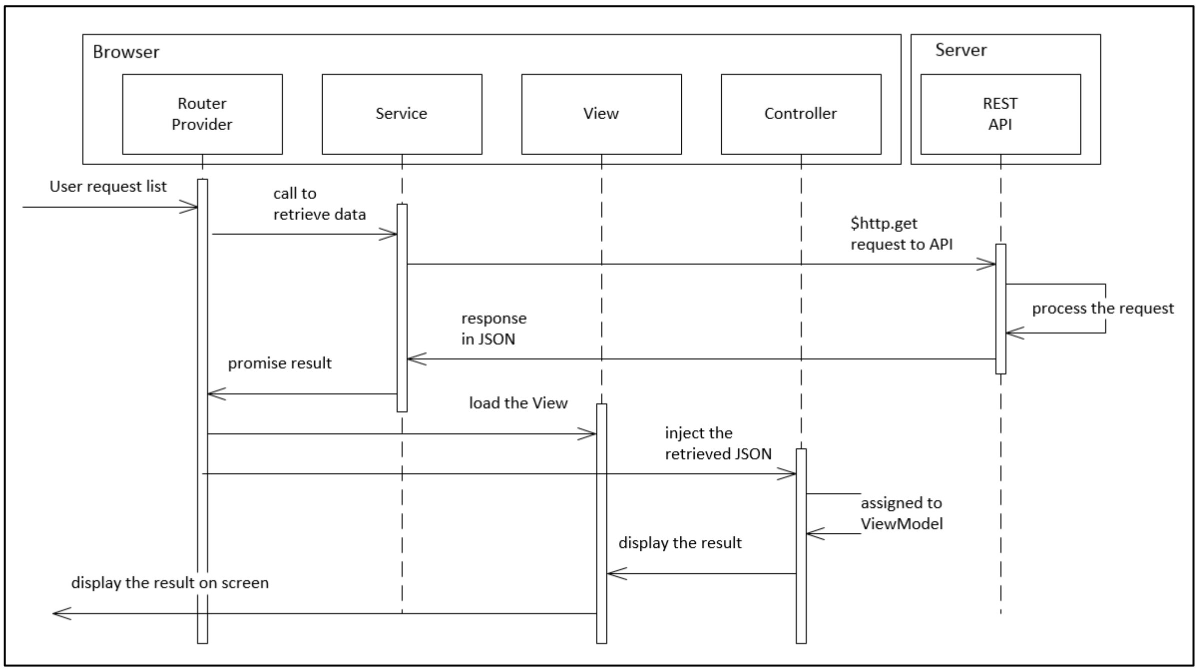


Figure 29: Overview of program flow on the client-side

Although the described program flow is the reading of data, creation or update of data follows the same way. In addition there is also validation of the data input in place. The data has to be validated on the server-side either way. However, the round trip to the server can be skipped if validation already happens on the client-side and the error is immediately presented to the user.

If there are errors on loading or processing the data the client will display an appropriate error message. The display handling of the errors is done in the “alertService”, which provides an application wide list of raised errors. This allows to raise errors from different components without tying them together, and even having the error information after changing state (and therefore changing controllers).

Because the software solution uses a custom locking approach (see 3.3.1 Multi-user capability, 4.1.6 Locking (concurrent editing) and 4.3.6 Locking (concurrent editing)) the data is always presented in a read-only view, and the user has to click on the “change”-Button. When the event for this button is raised, the client requests the data again, however, this time the client requests the data with a lock. This means that the REST-API on the server-side will try to lock the data for editing for the user only. The edit-view can only be presented if the lock is successfully created, otherwise it displays the user an error message describing that another user currently edits the requested data.

4.4.2 Server-side

The application server will be started with the startup of Tomcat, which automatically loads the web application. After the start the application will wait for requests from clients. Figure 30 shows the program flow of the server side.

The incoming request is first processed by the Apache Webserver, and if the request contains valid authentication information the request will be given to the Tomcat Webserver, which processes the request in the filter-chain (see 4.1.5 Authentication and Authorization). We add our authentication filter to the chain. Within this filter the authentication information (the user data and the roles of the user) are extracted and compared with already stored user data. If the user enters the application for the first time, a new user record is created. All further actions within the application are logged with the created user-id. Returning users will receive the same user-id as was created during their first visit. However, the role information is not stored in the application and will always be read during the request processing.

The URL-router of the web application will determine the REST resource class based on the URI-information of the request, and afterwards it will instantiate the class and call the method which matches the URI-information. The call will contain additional information like a JSON-payload (POST and PUT) or parameters like the id of an element (see 4.1.4 REST).

The “list” method in the resource class will issue a SQL-statement (written with jOOQ) on the DBMS. The result is a typed list of elements (record objects). Because we do not want to bind the client to ORM-details (among other reasons) we use our domain model objects. Therefore the record objects (in detail the values of those objects) have to be mapped to corresponding domain model objects (see 4.1.3 Persistence).

The result of the mapping is a typed list of domain model objects. To use the result on the client side we have to change the representation of those objects into a client readable format. In our application this format is JSON, which means we have to transform the domain model objects into JSON objects. Those JSON objects in combination with an HTTP-code will be sent to the client (see 4.1.4 REST).

Although this description is for the request of a list of objects, the handling of the request for editing data is very similar. The program flow will look like shown in Figure 30, but from the bottom to the top. The data will be received in JSON format and has to be translated into domain-model objects. Afterwards various validations are done, and the data will be mapped to a jOOQ record object, which will be used to persist the data in the DBMS.

Because we offer a REST-API we cannot trust the validity of the received data. Therefore typical validation steps on the server side are the check on mandatory fields (e.g. the name of an application country is mandatory) and unique constraints (e.g. should the name of an application country be unique among all application countries). Checks on dependent data (e.g. when persisting a patent application the request may contain an application country which is not available) are done by the underlying DBMS (e.g. foreign-key constraints).

The DBMS is used as data storage. The software solution does not use any custom programmed functions or stored procedures on the DBMS.

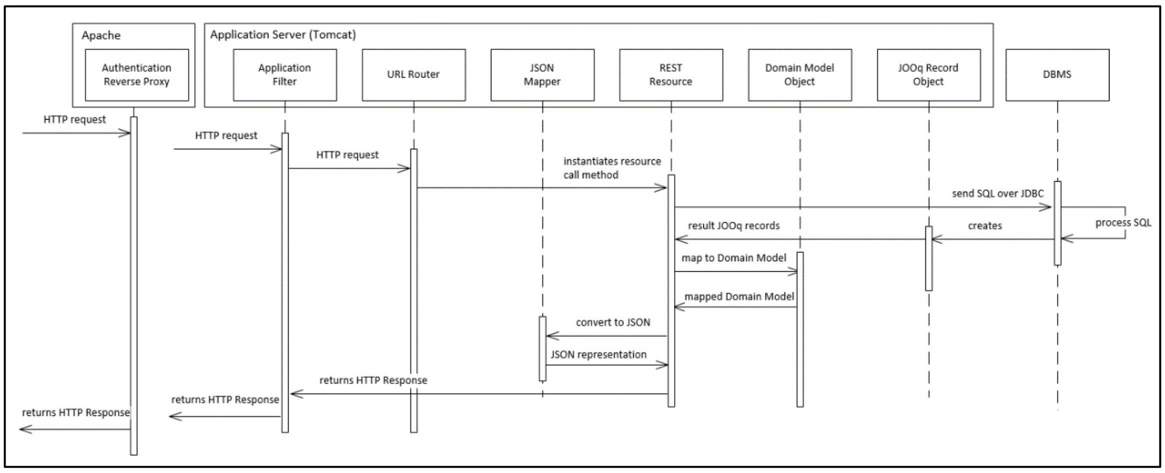


Figure 30: Overview of program flow on the server-side

5 Evaluation

Validation is “the process of evaluating software at the end of software development to ensure compliance with intended usage” [53]. In our solution we did not specify this criteria, although the validation is done by the end users who use the application and their feedback can be used as a result.

There are multiple solutions to evaluate the software solution. The most important fact is actual use of the application for the required purpose. Other evaluation options are the comparison with the Microsoft Excel based solution, a comparison with a commercial product and a comparison with the stated requirements. Software testing is also a form of evaluation.

5.1 Actual use of the software

Throughout the development project the end-users were part of the project team. During the early stages prototypes were shown on developer machines and exchanged via screenshots. After having a basic functionality (store an invention, a patent application and a granted patent) the prototype was installed on a test-server, where the end users had access to use the software with test data. Again feedback was given to the development team and with each iteration the feedback was used to change functionality, and the requirements list (see 3.2 Requirements elicitation) was used to add functionality.

When most of the required features were present the software application was installed on a server hosted by the university, and only accessible for authenticated users (see 4.1.5 Authentication and Authorization). Even if not fully finished the software could already be used to store the relevant data and the department of research support started immediately entering data into the system. Again useful feedback was provided, and changes and new features were added and installed on the server. While data entry was already possible on the test-system, the users still found required modifications through the process of data entry (live data).

Because of the early access and collaboration with the end users some of the required features were dropped (e.g. the e-mail notifications). On the other side new feature requirements, e.g. to store data about sale contracts and revenues, arose and were added. With the agile process of the project changes were instantly handled and increased the satisfaction of the end users. They had the direct opportunity to work on the project (which they did) and discuss every aspect of the solution.

As of the time of writing this thesis there were already dozens of inventions stored in the system, and multiple users used the application. Although during the initial data entry phase defects were found, those defects were not critical and there was no data loss or a defect where the users could not use the application at all.

The visually reduced user interface with only the necessary elements on the screens did not require any training for the participating end users. The vocabulary used in the views is taken from the requirements list, which in turn was written by the end users.

5.2 Comparison with the Microsoft Excel based Solution

When migrating from one solution to another solution a possible evaluation is the comparison of the data in both solutions. If the migration should not lose any data this can be done by comparing the objects and properties of both solutions.

This project is “based” on the Microsoft Excel solution, consisting of two sheets (see 2.2 JKU Excel solution) and the newly created software solution “invadmin”. Although the sheet organizes all information about one invention in one row, the application stores the information in various objects (or tables). Therefore the following comparison is done on the level of the objects of the application. Because the JKU-Excel is written in German, the column labels are presented in German.

In the following paragraphs we show that every column of the provided JKU-Excel solution has at least one corresponding property in the created software solution and therefore all data from the sheets can be stored within the application.

5.2.1 Invention

The invention is the main object in the solution. All other activities (like the application of a patent) are based on an invention. Table 14 lists the columns which are now stored in the invention object.

JKU-Excel column	invadmin property
Jahr	yearOfTheAnnouncement
Projektnummer	inventionNumber
Projektname	name
Erfinder + Gewichtung	The inventors and the share is stored in a separate object. Inventors are persons, which can be reused on other inventions.
JKU Erf / DM	inventionTypeId The inventions can have different types, which can be created or changed by the users. One type can be “JKU Erfindung”, another type can be “DM”.
Erfindungsmeldung	inventionNotificationPresent inventionNotificationReceived
Prüfung aws	awsOrderTypeId awsUploadOn

	There are different evaluations possible, the various types can be created or changed by the users.
Aufgriffsempfehlung	awsAcquisitionRecommendationId There are different recommendations possible, the various types can be created or changed by the users.
Ende Aufgriffsfrist	acceptanceDeadline acceptanceDeadlineOverriden The acceptance deadline is calculated automatically, however, the user may change the calculated value.
Aufgriff	acceptanceTypeId There are different types of acceptance possible, the various types can be created or changed by the users.
Kooperationspartner	cooperationPartnerTransferId cooperationPartnerName cooperationPartnertransferedOn There are different types of cooperation partner transfer possible, the various types can be created or changed by the users.

Table 14: Comparison of JKU-Excel columns with the invention object

5.2.2 Patent application

This object is used to store all data for patent application process. In the JKU-Excel solution there are multiple column sections for different patent applications. This was transformed into having multiple patent application objects connected to an invention. Table 15 lists the columns which are now stored in the patent application object.

JKU-Excel column	invadmin property
Prioritätsnummer Titel	name isPriorityApplication Every patent application has a name. In addition priority patent applications are marked.

Aktenzeich. PA	attorneyName attorneyFileReference
Anmeldetag	filedOn
Anmeldenr.	filedReference
Nationale Phase Frist	nationalizationDeadline
PCT-Anmeldung	applicationCountryId There are different types of application countries possible, the various types can be created or changed by the users.

Table 15: Comparison of JKU-Excel columns with the patent application object

5.2.3 Granted patent

This object is used to store all data for granted patents. In the JKU-Excel solution there are multiple column sections for different granted patents. This was transformed into having multiple granted patent objects connected to an invention and to a patent application. Table 16 lists the columns which are now stored in the granted patent object.

JKU-Excel column	invadmin property
AT-Patent Nr.	Number
Erteilungsdatum	publicationOn
Laufzeit	periodOfValidity
Erneuerung ab 6. Jahr	annualFeeDoneUntil stillValid

Table 16: Comparison of JKU-Excel columns with the granted patent object

5.2.4 Remaining columns

There are two remaining columns, “Verwertungserfolge” and “Info”. Both were also transferred into separate objects. The first one is saved within the objects saleContractor and revenue, the second one is stored in the infotext object.

5.3 Comparison with the requirements

At the beginning of the project a list of requirements were handed over to the development team (see 3.2 Requirements elicitation). As already described the document consisted of different parts.

The wish list contained various requirements which would be nice to be present in the final software solution. With the exception of a feature where the users can add extra fields to the objects all nice-to-have features are implemented.

The object and field list was used as a base for the various objects in the application, although during the various feedback cycles fields were removed, changed or new ones were added.

During the implementation phase various fields and features defined during the requirements elicitation phase (see 3.2 Requirements elicitation) were dropped. Table 17 lists the fields, features and a description of the reason why they were dropped.

Required field at the beginning	Reason for the drop
Erfindung - Aufteilung der Kosten	Can be done with the infotext object, which allows a categorization of the information stored within the object.
Erfindung – Verwertungserfolge	Can be done either with the infotext object or the saleContractor and revenue objects.
Felder für Prioritätsanmeldung	The complete “object” was dropped in favor of using a single patent application object with different options.
Patent – ist beendet + Grund	This information in detail is not necessary anymore. Instead of this field the patent status field can be used, where arbitrary patent status could be created as a reference list.
Dokumenten-Upload	This feature was dropped because the documents still have to reside on a file server and this will double the amount of work to store the documents.
Benachrichtigungsfunktion	This feature was dropped after seeing all relevant information within the software solution on the dashboard.

Table 17: List of dropped fields and features

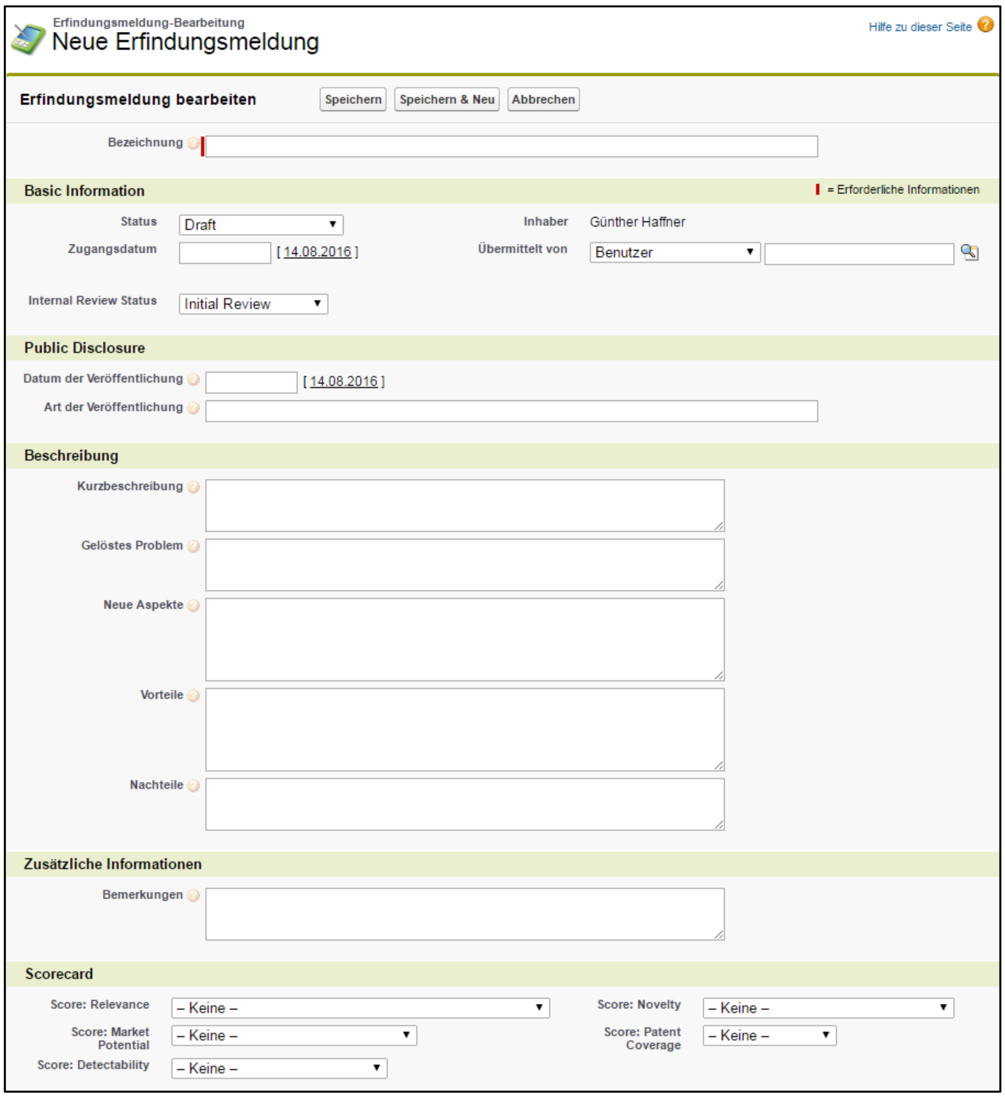
All other objects and fields from the requirements are present in the application. The required reports and lists can also be found within the application.

Because the end user already uses the software and entered information from various years we assume that the requirements are fulfilled with the delivered software solution.

5.4 Comparison with the commercial product

We requested a demo account for the product from IPfolio, which is also a web based tool for the administration of inventions, patent applications, patents and many more. The product is based on the salesforce platform, which provides the possibility to extend an application with custom objects and fields. [54] The platform is cloud based with various data centers around the globe.

With the demo account we were able to create an invention disclosure, which can be compared to the creation of an invention object in our solution, or with a new line in the JKU-Excel solution. Figure 31 shows the screen which is presented on the creation of a new invention object. Although not all required fields (from the JKU requirements) are displayed the product offers a feature called "total control", which allows the creation of custom fields. It means the fields posed in the requirements have to be searched within the solution, and if not found the fields have to be created.



Erfindungsmeldung-Bearbeitung

Neue Erfindungsmeldung

Hilfe zu dieser Seite

Erfindungsmeldung bearbeiten Speichern | Speichern & Neu | Abbrechen

Bezeichnung

Basic Information — = Erforderliche Informationen

Status	Draft	Inhaber	Günther Haffner
Zugangsdatum	[14.08.2016]	Übermittelt von	Benutzer
Internal Review Status	Initial Review		

Public Disclosure

Datum der Veröffentlichung	[14.08.2016]
Art der Veröffentlichung	

Beschreibung

Kurzbeschreibung

Gelöstes Problem

Neue Aspekte

Vorteile

Nachteile

Zusätzliche Informationen

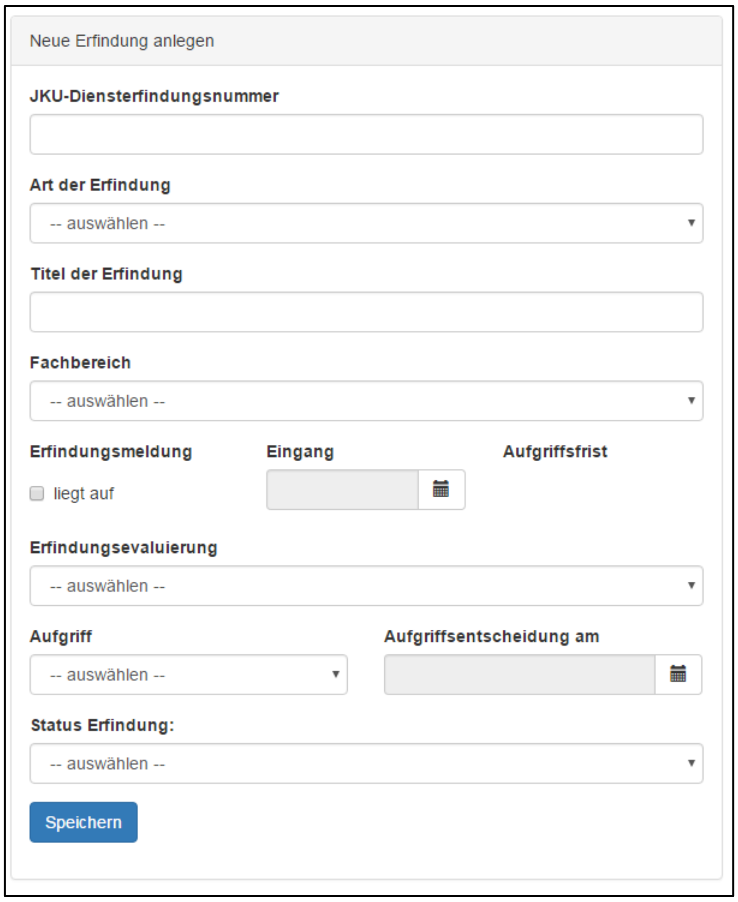
Bemerkungen

Scorecard

Score: Relevance	– Keine –	Score: Novelty	– Keine –
Score: Market Potential	– Keine –	Score: Patent Coverage	– Keine –
Score: Detectability	– Keine –		

Figure 31: Screenshot of IPfolio's new invention disclosure screen

In comparison Figure 32 shows a screenshot of the view where the user enters the details of the new invention in our software solution.


Neue Erfindung anlegen

JKU-Diensterfindungsnummer

Art der Erfindung: -- auswählen --

Titel der Erfindung

Fachbereich: -- auswählen --

Erfindungsmeldung: liegt auf

Eingang

Aufgriffsfrist

Erfindungsevaluierung: -- auswählen --

Aufgriff: -- auswählen --

Aufgriffsentscheidung am

Status Erfindung: -- auswählen --

Figure 32: Screenshot of invadmin’s new invention screen

All other objects from the posed requirements can also be found in the IPfolio solution, and after individually creating new fields all information can be stored within the solution. The application runs in the browser too, however, it is a classic Web application (see Figure 4) where the page is completely loaded while various operations are performed.

Our software solution does fulfill the requirements from the department of research support as it is, whereas the IPfolio solution first has to be adopted to be usable for the end users. Because the data is stored in the cloud the university needs to decide if it wants to store such sensitive data in a cloud. IPfolio’s rich feature set, on top of the salesforce features, allows the customization even on the view level, which makes it possible to create the same structure like the structure we have created with our domain model and application.

We deliver out of the box reports, and allow the export of the relevant data. IPfolio also delivers reports, although they do not contain the structure defined in the requirements. Salesforce offers a report builder, where the end user can create report-templates and add them to the application.

The major difference for the university between the software from IPfolio and our solution is the price of the solution. Next to the price of the solution itself (team edition starts with € 1.000 per month) the user will require training (because of the feature-rich application) and additional set up (like customizing). The process of customizing is not trivial and requires extensive training. Or the company does the customizing itself, which will not be for free at all. Our solution does not pose extra costs for the university. The clear and simple interface allows the use without training. Only the processes and vocabulary have to be learned by new users. The university has the source code of the solution and can therefore create new releases whenever necessary.

5.5 Testing

Software testing can be applied to the software solution. Although tests can show that the intended use executes as expected they cannot show that there are no defects (the absence of defects). The process of testing can be seen as validation and verification.

There are different kinds of test methods available but we do not want to describe testing in detail. However, we want to provide information about the testing that occurred during the development and test cases used in the solution for future releases of the software.

Because we mainly load data from or store data in the database, transforming the objects from one representation to another one, there are no algorithmic challenges in the web application server. Therefore we did not write Unit-Tests.

The testing of the user interface was done by using a browser and entering the information, saving and retrieving the saved state from the application. Typically web applications can be tested with Selenium, which allows the scripted execution of test-cases and therefore the automation of testing. [55] However, Angular and other MV* frameworks dynamically load HTML-templates and Selenium cannot be used on applications based on those frameworks. There is the tool Protractor which allows the testing of Angular applications. [56] However, we did not write test cases for this tool because we did the testing manually. Although we recommend to use the tool in the future.

The decoupled design with the REST-API allows the testing of the API with common available testing tools. We decided to use frisby.js, because it is available for free and the test cases can be written in JavaScript. [57] The library allows us to use our REST-API from JavaScript without using a browser at all. The test cases run in parallel, which also allows the creation of more load on the application server than by doing manual testing with the browser. A test case can issue requests and wait for the response. The response can be analyzed, e.g. the HTTP-code could be verified, or the JSON-structure. It is even possible to compare the received JSON with a predefined JSON.

When writing the test cases we tried to find defects on the one side, and on the other side tried to raise confidence that the application server fulfills the requests like intended. We created for every domain model which is used as data store an individual test file, which contain all test-cases for that domain model.

As an example, the test cases from the application country are described in Table 18. The test case is not named, it is always a short description of what the test case should do. This description is also used in displaying the results of a test run.

Test case	Description
It should get application countries	The list of application countries is requested from the REST-Endpoint. The case verifies ▪ the HTTP-Code (expected is the code 200), ▪ the HTTP-Header, which should contain the content-type "application/json", ▪ the JSON-result, which should be the property "objects" from the type "Array", ▪ the JSON-result of the property "objects" which should have the properties "id" ("Number"), "name" ("String"), "active" ("Boolean").
It should get one application country by id	After the list of application countries was retrieved the first id is selected (by the previous test case) and this case uses that id to request a single application country from the server. The case verifies ▪ the HTTP-Code (expected is the code 200), ▪ the HTTP-Header, which should contain the content-type "application/json", ▪ the JSON-result, which should have the properties "id" ("Number"), "name" ("String"), "active" ("Boolean").
It should insert an application country	This test case creates the JSON for an application country and "post" this to the REST-Endpoint. Because this test case creates an application country we expect the HTTP-code 201, and the result of the call is a JSON object containing the property "details", which is in fact the identifier (id) of the newly created object.

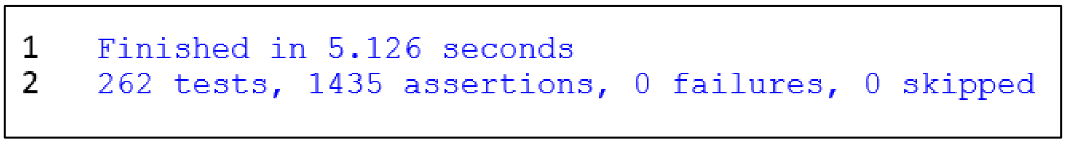
It should not insert an application country (same name exists)	This test case starts with the same data as the previous one. Because names of elements should be unique this request needs to be rejected by the server. Therefore we expect the HTTP-code 406.
It should update an application country	This test case changes the data of the previous created application country. Therefore we have extracted the id of the result and use this id to create the URL for the change. After successful change on the server-side we expect the HTTP-code 200.
It should not update the application country (missing mandatory fields)	This test case should fail to update the data on the server because we send only data without mandatory fields. We expect again the HTTP-code 406.
it should not update the application country (wrong id in the payload)	This test case should also fail to update the data on the server. The reason is, that the payload (the JSON-data send to the server) may also contain the identifier (id) of the object, and this id may be different from the id presented in the URL. Although the server application can just use the information from the URL the API expects valid data, which is not the case. Therefore we expect the HTTP-code 406.
it should get the updated application country by id	This test case requests the previous created and updated application country and verifies again if the data is the same as expected.
it should delete the inserted / updated application country by id	Although the software solution does not offer to delete the application countries in the user interface, the REST-API does offer this functionality (which in turn is convenient during testing).
It should not find the deleted application country by id	After deletion of the application country another delete request should fail with the expected Http-code 404.

Table 18: Test cases for API test of application country

The test cases are executed in parallel, however, some of the test cases depend on other cases and are only started after the previous test case finished. We provide multiple files with test cases, and the cases are always only chained within a file, that means that still a lot of test cases are run in parallel.

Because of the test automation the test cases can be run anytime. However, not all objects can be deleted, and the database will grow with the created elements. Therefore we provide a SQL-statement, which creates ordered delete statements for all tables of the solution. Each delete statement has a constraint on the identifier. Before running the test cases, the SQL-statement should be executed and the result should be saved for execution after the test cases run. All data inserted after the initial run of the statement will be removed from the database. It is worth mentioning that during the automatic testing no user should interact with the software, otherwise the data may be lost after executing the SQL-statement to delete the created data.

When changing the software solution the test cases may also change, and as a good practice we recommend to change them immediately. We did verify that the expected results are delivered when using the REST-API. Figure 33 shows the result of the automation test run.



```
Finished in 5.126 seconds
262 tests, 1435 assertions, 0 failures, 0 skipped
```

Figure 33: frisby.js API-Test result

6 Summary

6.1 The good and the bad of the project

Software development of is not always frictionless. In this project the requirements were described on the one side very short and superficial (e.g. login with university login), on the other hand very detailed, like the provided field list.

With the collaboration from the beginning, first with screenshot exchange, later by using the test-system, the end users were included in the project during the whole development process. This helped to avoid misunderstandings in every phase and the initial design decisions did not have to be changed. The project team provided enough time to work on the project and delivered necessary inputs in a timely manner. On the other side the development team delivered a solution tailored to the requirements from the end users, and the satisfaction of the end users with the delivered product is very high.

The project also depended on information from the universities IT management. During the early stages the information for the university login were requested. The first response came immediately and named a person responsible for the login. We tried to reach that person, which was not possible during a duration of two months. During this time our application grew and each week additional features were added. We decided to wait for the implementation of the authentication and authorization to the very last moment. This decision was not easy to make, but without information from the university we could not implement that part in a way to satisfy the end users. At that time we included a standard authentication mechanism as part of Tomcat and decided to change that fact when information is available.

When the application was ready to be used and data from the past years could be entered we still had no information about the login, and also no information about the target platform the software should be installed. We declined the proposal from the university to first install the operation system and then the software by ourselves on one of the provided virtual machines. We provide the application, but the application will run on the server the next years and someone has to be responsible for the server. In our opinion this person should provide the installation of the server and necessary components for our software. Because it is a web application with a database backend the installation is just the creation of a database and one time execution of the creation SQL script (which creates the schema), and the installation of the web application is just copying the files in the application folder. Finally the configuration of the application has to be adopted.

The university finally offered the resources to host the application on one of their servers and even provide the installed system in a state where we only had to install our software. The responsible employee also provided the information for using the Single Sign On and suggested a solution to integrate that with our application.

6.2 Development problems

The software architecture and the chosen programming languages and frameworks needed a lot of learning and knowledge gathering to provide the final application. Today's software development is often characterized with the use of various components from other organizations or even other individuals.

JavaScript is a script language, which provides basic object oriented features. It is weakly-typed, which means that variables can have every type and even change the type, the type of a variable is not declared at all. This allows flexible solutions, but opens up various problems on the other side. The biggest one is the missing compile time check. With Java, e.g., we compile a program and if we have type or typo errors the compile will complain and abort the compilation process. We do not get a compiled program and therefore we have to fix those errors before creating a runnable program. With JavaScript the code gets first executed in the browser. Because variables do not need to be declared, a typo often leads to defects in the software which has to be traced and fixed.

The component orientation of the frameworks used allows the extension with other components (or libraries). Those components may also contain defects which may affect in our software solution. Therefore the components have to be chosen wisely. Sometimes those components do not work if used together, or have other known limitations and we have to decide if the limitation is acceptable or if we have to use another component, e.g. it is recommended to disable the Angular debug mode on production systems, but it is currently not possible in our solution because we use Angular-datatables which have do not work as expected when the debug mode is disabled [58].

The frameworks and the external components are frequently updated and new versions get available. During the development we upgraded the application from Angular 1.3 (which was the version the web template was generated with) to Angular 1.5.6. Not every component was available for that Angular version. In such a case it will be necessary to wait for an update of the component, wait in general for updating the framework, or change to another component which will work with the new version. Although the update process can be done automatically with bower, testing and adoption of the application will still be necessary.

REST-APIs can be found on many places in the Internet, and they are easily developed. However, thinking in resources instead of domain models has to be done when designing the API. As long as only our own application uses the API we can match up the client- and the server-side. But we should also think of future users of the API, and they should be able to use the API without knowing the details we implemented on our own client.

The decision not to make a complex domain model was always under consideration when the client needs the details of all associated objects which are just provided as id within the main object. Most of the time the client also needs all elements of a reference, and not only the one which is currently stored. This shows that the decision of using the simple domain model was correct, even if it requires often multiple requests from the client to the API to provide the necessary user experience.

6.3 Potential extension

The provided software application is in release version 1.0. Because the source code is provided the application can be fixed (if a defect is found), modified or extended with new fields or even new features. We propose the following future enhancements and changes of the software solution:

- *Automated UI-Testing:* With the help of Protractor those test cases can be created and used to test the software after every change.
- *AngularJS 2.0:* There is an upcoming major release of AngularJS. With this release applications are more constructed like components and the developers behind AngularJS changed a lot of the framework.
- *TypeScript:* The client-side (services, controller etc) should be migrated to TypeScript instead of JavaScript. TypeScript adds type support and object orientation on top of JavaScript and can be compiled (before releasing the solution) to plain JavaScript [59]. This will help to circumvent problems described in the previous section.
- *Deletion of elements:* Although during the project we decided to not allow the deletion of data through the user it should be added to a future release. This will give the users the flexibility to remove data they entered accidentally.
- *Caching of data on the client:* There are multiple reference lists used. They change not very often, but are not cached on the server- and also not on the client-side. Most of the views need the data from reference lists, hence they are always loaded from the server. With a client side caching mechanism the data will be available for all views and not loaded each time it is used. This will speed up the client and the server (even if the performance is fast enough at the moment). Websockets can be used to notify the clients if the cached data changed.
- *Reporting framework:* Currently the reports are created from the JSON-data received from the REST-API. The data has to be transformed into matrix form to be usable for AngularJS directives to render the result. We suggest to evaluate the usage of a reporting framework like JasperReports Server [60]. The report creation starts on top of the database (not the REST-API) and is a dedicated solution for creation and rendering of reports. Adding or modifying reports does not change our software solution at all.
- *Multi-tenant-capability:* The provided application may also be used by other Austrian universities. The software can be changed to allow the usage with tenant information. This can be done with an additional column on all tables, or e.g. with a separate schema (which is a feature of jOOQ). The alternative is to create individual installations for different universities. Nevertheless, the other users may pose new requirements that differ from the requirements of the JKU. For that case a customization concept has to be created.
- *Invention disclosure form:* Currently the process at the JKU involves a written form submitted to the department of research support. This could be enhanced by providing

an online form where the data could be entered. A process to add this data to the application’s inventions has to be defined, e.g. a manual check of the submitted data.

- *Improving Search:* Currently the global search is implemented by looking at the various textual fields in the database with an SQL “like” search which is basically doing a string comparison. Because the database does not contain that much information this type of search can be performed fast. However, during the life time of the application the data will be increase, and the search performance will decrease. There are at least two possible solutions for improving the search: the usage of full text indices from PostgreSQL or the usage of a dedicated search library like Apache Lucene.

Bibliography

[1] E. C. Gerhart, Quote It Completely: World Reference Guide to More Than 5,500 Memorable Quotations from Law and Literature, Getzville, US: William s Hein & Company, 1998.

[2] 2016-07. [Online]. Available: https://shibboleth.net/.

[3] FUN, "JKU Website," JKU, 2015-04-09. [Online]. Available: http://www.jku.at/content/e263/e257286/e258292/e258324. [Accessed 2016-07].

[4] L. J. Heinrich and D. Stelzer, Informationsmanagement, 10th ed., Munich, GER, 2011, p. 273.

[5] D. Closa, A. Gardiner, F. Giemsa and J. Machek, Patent Law for Computer Scientists, Heidelberg, GER: Springer-Verlag, 2010, p. 14.

[6] O. Offenburger, Patent und Patentrecherche, Wiesbaden, GER, 2014, p. 36.

[7] D. Closa, A. Gardiner, F. Giemsa and J. Machek, Patent Law for Computer Scientists, Heidelberg, GER: Springer-Verlag, 2010, p. 4.

[8] EPO, "EPO," 2015-08-03. [Online]. Available: http://www.epo.org/applying/european.html. [Accessed 2016-07].

[9] W. Hahnl, Praktische Methoden des Erfindens, Heidelberg, GER, 2015, p. 51ff.

[10] JKU, *IPR-Strategie der JKU Grundsätze der Dissemination und Unterstützung der Verwertung von JKU Forschungsergebnissen (Version 2),* Linz, AT, 2015-03-01.

[11] *§6f Patentgesetz 1970 BGBl. 1970/259 idF BGBl. I 2013/126 (PatG).*

[12] aws, "aws Marktrecherche," [Online]. Available: http://www.awsg.at/Content.Node/foerderungen_alle/patentservice/48303.php. [Accessed 2016-07].

[13] L. J. Heinrich and D. Stelzer, Informationsmanagement, 10th ed., Munich, GER, 2011, p. 249.

[14] P. Mertens and e. al., Grundzüge der Wirtschaftsinformatik, 11th ed., Heidelberg, GER, 2012, p. 137ff.

[15] "ANAQUA Unified IP management for corporations," [Online]. Available: http://www.anaqua.com/corporate/products. [Accessed 2016-07].

[16] "ANAQUA Our Story," [Online]. Available: http://www.anaqua.com/about-us/our-story. [Accessed 2016-07].

[17] I. corporation, "IPfolio Manage your IP portfolio strategically," [Online]. Available: http://www.ipfolio.com/our-product/ip-manager/. [Accessed 2016-07].

[18] H. W. Wieczorrek and P. Mertens, Management von IT-Projekten, Heidelberg, GER, 2011, p. 84ff.

[19] K. Conboy and B. Fitzgerald, "Toward a Conceptual Framework of Agile Methods," in *XP/Agile Universe 2004*, Calgary, CA, 2004.

[20] E. Hull, K. Jackson and J. Dick, Requirements Engineering, 3rd ed., London, UK, 2011, p. 2.

[21] D. E. Perry and A. L. Wolf, "Foundations for the Study of Software Architecture," *Software Engineering Notes,* vol. 17, no. 4, pp. 40-52, 1992.

[22] P. B. Kruchten, The Rational Unified Process, vol. 3rd, Addison-Wesley Professional, 2003.

[23] M. Fowler, Patterns of Enterprise Application Architecture, New Jersey, US, 2014.

[24] P. Mertens and e. al., Grundzüge der Wirtschaftsinformatik, 11th ed., Heidelberg, GER, 2012, p. 28.

[25] "Oracle The Java EE 5 Tutorial - What is a servlet?," [Online]. Available: http://docs.oracle.com/javaee/5/tutorial/doc/bnafe.html. [Accessed 2016-07].

[26] "Oracle JavaServer Pages Technology - Frequently Asked Questions," [Online]. Available: http://www.oracle.com/technetwork/java/faq-137059.html. [Accessed 2016-07].

[27] "Oracle Introduction to Javaserver Faces," [Online]. Available: http://www.oracle.com/technetwork/topics/index-090910.html. [Accessed 2016-07].

[28] F. Petitit and M. Tricot, "The new Web application architectures and their impacts for enterprises – Part 1," OCTO, 2014-03-21. [Online]. Available: http://blog.octo.com/en/new-web-application-architectures-and-impacts-for-enterprises-1/. [Accessed 2016-07].

[29] W3C, "HTML5 - A vocabulary and associated APIs for HTML and XHTML," 2014-10. [Online]. Available: https://www.w3.org/TR/html5/. [Accessed 2016-08].

[30] Twitter, "Bootstrap," [Online]. Available: http://getbootstrap.com/. [Accessed 2016-07].

[31] T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2," 2008-08. [Online]. Available: https://tools.ietf.org/html/rfc5246. [Accessed 2016-07].

[32] H. Schuldt, "Multi-Tier Architecture," in *Encyclopedia of Database Systems*, New York, US, 2009, p. 1862–1864.

[33] "jOOQ," Data Geekery GmbH, 2016. [Online]. Available: http://www.jooq.org/. [Accessed 2016-07].

[34] M. Inden, Der Java-Profi: Persistenzlösungen und REST-Services, Heidelberg, GER, 2016, p. 269ff.

[35] R. Fiedling and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content," IETF, 2014-06. [Online]. Available: https://tools.ietf.org/html/rfc7231. [Accessed 2016-08].

[36] S. Newman, Building Microservices, 4th ed., Sebastopol, USA, 2015, p. 215f.

[37] Fielding and e. al., "Method definitions of RFC 2616," [Online]. Available: https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html. [Accessed 2016-07].

[38] "JSR 339: JAX-RS 2.0: The Java API for RESTful Web Services," Oracle, 2014-10-03. [Online]. Available: https://jcp.org/en/jsr/detail?id=339. [Accessed 2016-07].

[39] "Jersey RESTful Web Services in Java," Oracle, 2016-06-09. [Online]. Available: https://jersey.java.net/. [Accessed 2016-07].

[40] N. Freed, J. Klensin and T. Hansen, "Media Type Specifications and Registration Procedures," 2013-01. [Online]. Available: https://tools.ietf.org/html/rfc6838. [Accessed 2016-08].

[41] D. Salomon, Elements of Computer Security, New York, USA, 2010, pp. 221, 332.

[42] "Apache Tomcat 8 Documentation - Realm Configuration HOW-TO," 2016-07. [Online]. Available: https://tomcat.apache.org/tomcat-8.0-doc/realm-howto.html. [Accessed 2016-08].

[43] O. Consortium, "Project License," 2016-03-05. [Online]. Available: http://asm.ow2.org/license.html. [Accessed 2016-07].

[44] S. Abiteboul, R. Hull and V. Vianu, Foundations of Databases, Addison-Wesley Publishing Company, 1995.

[45] "Angular.js Developer Guide," Google, [Online]. Available: https://docs.angularjs.org/guide. [Accessed 2016-07].

[46] "Angularjs architecture overview," [Online]. Available: http://tutorialspointexamples.com/angularjs-architecture-overview-core-concepts-advantages-disadvantages/. [Accessed 2016-07].

[47] S. Sehsadri and B. Green, AngularJS Up & Running, Sebastopol, USA, 2014.

[48] Microsoft, "The MVVM Pattern," 2012-02-10. [Online]. Available: https://msdn.microsoft.com/en-us/library/hh848246.aspx. [Accessed 2016-07].

[49] "Learning JavaScript Design Patterns - MVVM," [Online]. Available: https://www.safaribooksonline.com/library/view/learning-javascript-design/9781449334840/ch10s06.html. [Accessed 2016-07].

[50] J. Papa, "AngularJS's Controller As and the vm Variable," 2014-06-24. [Online]. Available: https://johnpapa.net/angularjss-controller-as-and-the-vm-variable/. [Accessed 2016-07].

[51] "Font Awesome - The iconic font and CSS toolkit," [Online]. Available: http://fontawesome.io/. [Accessed 2016-08].

[52] E. Gamma, R. Helm, R. Johnson and J. Vlissides, Design Patterns - Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995, p. 127ff.

[53] P. Ammann and J. Offutt, Introduction to Software Testing, Cambridge, UK, 2008, p. 11ff.

[54] i. salesforce.com, "salesforce," [Online]. Available: https://www.salesforce.com. [Accessed 2016-08].

[55] "Selenium Browser Automation," [Online]. Available: http://www.seleniumhq.org/. [Accessed 2016-08].

[56] "Protactor - end to end testing for AngularJS," [Online]. Available: http://www.protractortest.org/.

[57] L. Vance, "REST API Testing," 2014. [Online]. Available: http://frisbyjs.com/. [Accessed 2016-08].

[58] "angular-datatables issues," 2016. [Online]. Available: https://github.com/l-lin/angular-datatables/issues/746. [Accessed 2016-08].

[59] Microsoft, "TypeScript," 2016. [Online]. Available: https://www.typescriptlang.org/. [Accessed 2016-08].

[60] I. TIBCO Software, "Jaspersoft Community," 2016. [Online]. Available: http://community.jaspersoft.com/. [Accessed 2016-08].

Appendix

User documentation

Because the software user interface is completely in German language the user documentation is provided in German language as a separate document on the attached CD.

Installation procedure for live system

The following procedure does not include details about the setup of the integration with Shibboleth. This procedure can be used to setup the software after the initial setup of the prerequisites from the IT department.

Prerequisites:

- Apache HTTP server for authentication (version depending on the Shibboleth integration)
- Tomcat web server for the application (Tomcat version 7 or newer)
- Java JVM 1.8
- PostgreSQL (tested with version 9.2 or newer)
- PostgreSQL database and user with full access to that database

The PostgreSQL and Tomcat do not have to be on the same server.

Installation:

- On the destination database the schema has to be created. This can be done with the script “invadmin-schema.sql”. Although the schema may change in the future the developer has to provide schema upgrade files (for existing installations) and should change the installation script for future installations.
- The Java application should be extracted (if it is a WAR) and the files should be copied to the root directory of the application on the Tomcat server.
- The AngularJS application should be copied to the same location (and may replace files if necessary).

Configuration

- *Database Access:* In the folder WEB-INF/classes the file **database.properties** has to be modified.
- *Logging:* In the folder WEB-INF/classes the file **log4j2.xml** has to be modified. At least the path where the log files should be placed has to be modified.
- *REST-API-Url:* In the folder scripts the file **config.js** has to be modified. The URL of the REST-API-Endpoint (which is the same as the web application URL, but extended

with “rest”) has to be modified two times, because the start page also needs that information before the AngularJS application is instantiated.

Availability check

After successfully copying all files, creating the schema and modification of the configuration files an availability check can be made. A request to the resource “hello.jsp” will display a page with the server time and confirms that the JSP servlets will work. A request to the resource “rest/hello” will use the REST-API and should display “hello, this was resty yeah!!” and confirms the availability of the REST-API.

Installation procedure for developer system

Prerequisites:

- Node JS (https://nodejs.org) (current version, developed with 6.2)
- npm (which is at the time being included in Node JS)
- Git client (https://git-scm.com/) (current version, developed with 2.8.1)
- Java JDK (current version, developed with Java 8 (1.8.0.05)
- Tomcat (https://tomcat.apache.org/) (current version, developed with 8.0, installed live system with version 7.0)
- Eclipse (https://eclipse.org) (current version, developed with 4.4.2)
- PostgreSQL (https://www.postgresql.org/) (current version, developed with 9.3)

Basically the solution consists of two separate applications: the Java Web Application (developed within Eclipse and hosted with Tomcat) and the Angular Client Application (developed within a JavaScript Editor and hosted with Node).

Installation Database:

After installation of PostgreSQL a database for the application has to be created. A database user / role for access to this database has to be created. The name of the database, the login and the password are needed to configure the Web Application.

The database schema has to be created. Therefore, the file “invadmin-schema.sql” has to be used and the content has to be executed with “psql” or another tool. Although the schema may change in the future the developer has to provide schema upgrade files (for existing installations) and should change the installation script for future installations.

Installation Web Application:

The web was developed within Eclipse, and can be loaded. To run the web application inside Eclipse it is necessary to configure Eclipse and the web application to use the installed Tomcat Server.

When the application is loaded within Eclipse, there is on the bottom of the Eclipse Application a tab called “Servers”. There is the link to create a new server connection. The installed Tomcat Version has to be chosen, and in the next dialog the invadmin-application has to be added to the “Configured” area.

As already described in the previous section the application needs to be configured. In Eclipse this can be done on the left side in the Project Explorer. The node “Java Resources” contains a folder “config” where the configuration files are placed. They need to be adopted to the settings of the developer environment.

Installation Angular Application:

The Angular application consists of static files. They can be served by various HTTP servers, even Tomcat could be used. However, for better support during development the application will be served by node. The installation of the angular application is putting all files on the hard drive of the development workstation.

For various tasks on the application source the grunt task-runner is used. The installation of the grunt task-runner is:

```
npm install –g grunt-cli
```

For the management of third party libraries the package manager bower is used. The installation of bower is:

```
npm install –g bower
```

After both are installed, all components can be downloaded. This is done by changing into the directory where the file “Gruntfile.js” is located, and then the following command has to be executed:

```
npm install
```

After retrieving all dependent libraries (both for the web application itself and for the npm system) the Angular application can be started with the command:

```
npm start
```

This will start a node web server on port 900 and will automatically open the default browser with the URL of the application. The start script (npm start) also contains a module to watch on the application files and reload the page in the browser if files change.

The angular application may need a change of the configuration for accessing the REST service. This configuration has to be done in the file “config.js” (folder scripts).

Testing the API:

The test cases are stored in the folder “testAPI/spec” and can be executed with the command

```
jasmine-node spec
```

to execute all cases stored in the folder. Single cases can be executed with the following command:

```
jasmine-node spec\01-currentUser-spec.js
```

It is advised to use the SQL script “beforeTestCreateDeleteStatements.sql” before running the test cases. The script will create the delete statements to remove all data created by the test cases.

Update of Schema

The schema of the database can be updated by executing SQL scripts. The update should always be kept within the application source (as sql file). The script for initial creation of the schema should also be adopted if there are schema changes. In that way a future installation can be done with that script.

After updating the schema in the developer database it is necessary to update the generated Java classes (jOOQ). Therefore the batch file “genclassesExt.bat” (Java application, folder “DbTools”) can be used. The configuration (database access) has to be done in the file “config.xml” of the same folder.

Eclipse does not automatically refresh the files if they were changed. After running the generation-script the files in Eclipse have to be refreshed (Project Explorer, Refresh).

Creation of a release

A release of the application consists of three parts:

- Database schema changes
- Java Web application
- Angular application

The database schema changes have to be SQL scripts which can be executed.

The Java Web application for release can be created by doing a WAR export within Eclipse. The file has to be extracted. If the release is for update, the configuration files have to be removed. Otherwise the configuration files should be changed (they contain at least one password).

The Angular application for release can be created by executing the following command (in the folder the application resides):

```
npm run dist
```

This will create (or empty) a folder named “dist” and copies all relevant files for the release into this directory. If the release is for update, the configuration file has to be removed.

There is a batch file “prepareUpdate.cmd” included which can be used to combine the files of the Java application and the files of the Angular application and to remove the configuration files.

Update of the system

It is recommended to stop the Tomcat service before doing the update. This will prohibit users from accessing the application during the update. The database scripts can be executed on the installed PostgreSQL server.

The files both from the Java application and from the Angular application have to be copied to the application path on the server.

Database schema

The database schema is shown on an extra page (next page).

Backup and Restore

The backup process of the application consists of two independent tasks: the backup of the database and the backup of the file system.

To backup the PostgreSQL database there are various methods available. “pg_dump” creates a text file which can be used to recreate the database and the content of the database. This method can be used when the database service is in running state.

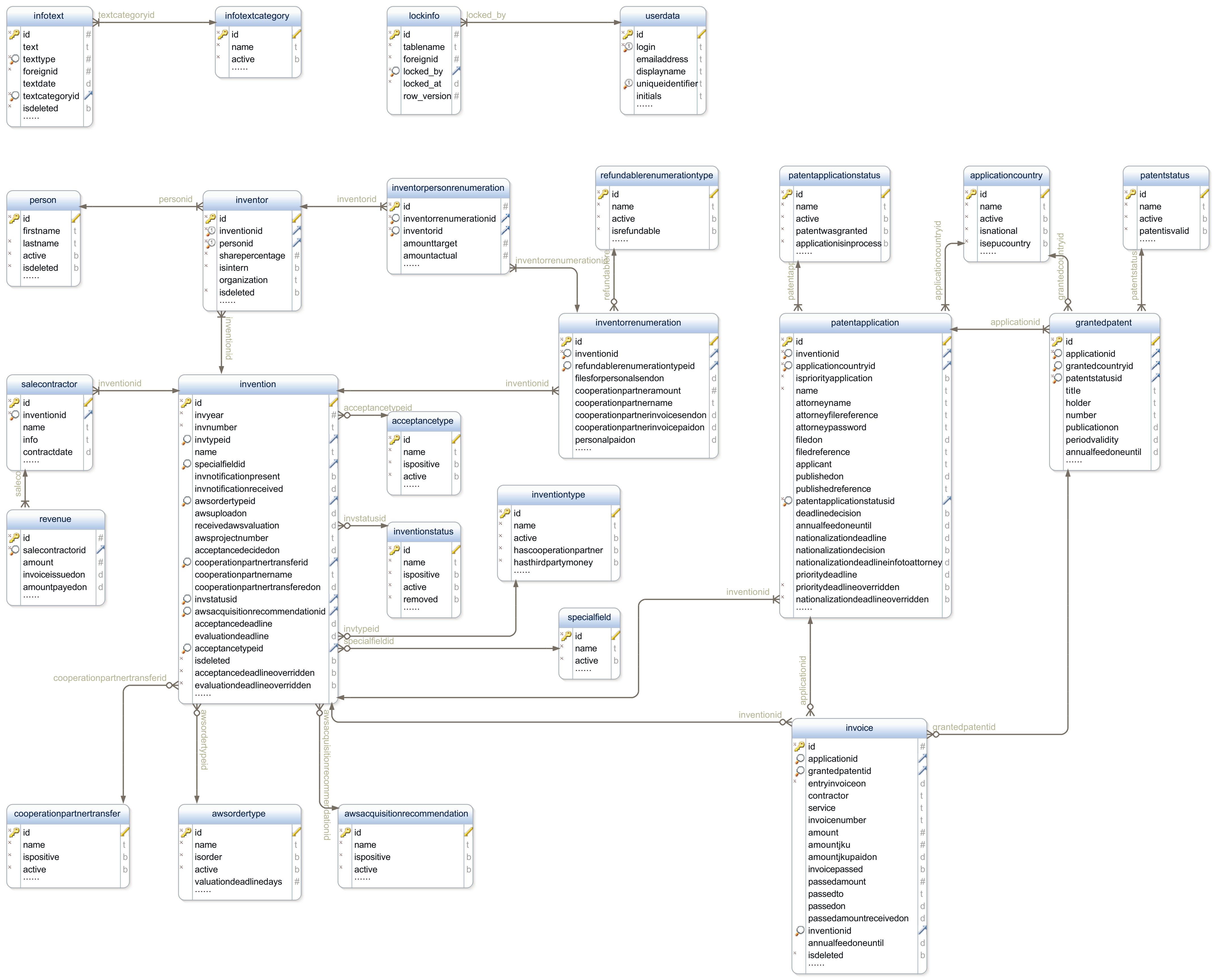
Another method is the file system level backup. This method copies the files which are used to store the data. However, this method is only usable if the database service is stopped. This could be done during non-office-hours.

The backup of the file system is just a copy process, where all necessary files are copied to a backup location. The files are typically within the Tomcat “webapps” folder. It is recommended to backup also the configuration files from Tomcat and die log files from the application.

The restore process is slightly the same, but in the other direction. Restoring the dump-file is possible with the command line tool “psql”, restoring the application files is by copying them to the previous backuped location.

Description of Log-Files

For logging log4j is used. The logging can be configured with the log4j.xml file. Each class uses a logger named after the class. However, there are currently three loggers configured.


infotext
id
text
texttype
foreignid
textdate
textcategoryid
isdeleted
textcategoryid
infotextcategory
id
name
active
lockinfo
id
tablename
foreignid
locked_by
locked_at
row_version
locked_by
userdata
id
login
emailaddress
displayname
uniqueidentifier
initials
person
id
firstname
lastname
active
isdeleted
personid
inventor
id
inventionid
personid
sharepercentage
isintern
organization
isdeleted
inventorid
inventorpersonrenumeration
id
inventorrenumerationid
inventorid
amounttarget
amountactual
inventorrenumerationid
refundablerenumerationtype
id
name
active
isrefundable
refundable
patentapplicationstatus
id
name
active
patentwasgranted
applicationisinprocess
patentapp
applicationcountry
id
name
active
isnational
isepucountry
applicationcountryid
grantedcountryid
patentstatus
id
name
active
patentisvalid
patentstatus
inventorrenumeration
id
inventionid
refundablerenumerationtypeid
filesforpersonalsendon
cooperationpartneramount
cooperationpartnername
cooperationpartnerinvoicesendon
cooperationpartnerinvoicepaidon
personalpaidon
patentapplication
id
inventionid
applicationcountryid
ispriorityapplication
name
attorneyname
attorneyfilereference
attorneypassword
filedon
filedreference
applicant
publishedon
publishedreference
patentapplicationstatusid
deadlinedecision
annualfeedoneuntil
nationalizationdeadline
nationalizationdecision
nationalizationdeadlineinfotoattorney
prioritydeadline
prioritydeadlineoverridden
nationalizationdeadlineoverridden
applicationid
grantedpatent
id
applicationid
grantedcountryid
patentstatusid
title
holder
number
publicationon
periodvalidity
annualfeedoneuntil
salecontractor
id
inventionid
name
info
contractdate
inventionid
saleco
revenue
id
salecontractorid
amount
invoiceissuedon
amountpayedon
inventionid
invention
id
invyear
invnumber
invtypeid
name
specialfieldid
invnotificationpresent
invnotificationreceived
awsordertypeid
awsuploadon
receivedawsvaluation
awsprojectnumber
acceptancedecidedon
cooperationpartnertransferid
cooperationpartnername
cooperationpartnertransferedon
invstatusid
awsacquisitionrecommendationid
acceptancedeadline
evaluationdeadline
acceptancetypeid
isdeleted
acceptancedeadlineoverridden
evaluationdeadlineoverridden
inventionid
acceptancetypeid
acceptancetype
id
name
ispositive
active
invstatusid
inventionstatus
id
name
ispositive
active
removed
inventiontype
id
name
active
hascooperationpartner
hasthirdpartymoney
invtypeid
specialfieldid
specialfield
id
name
active
inventionid
cooperationpartnertransferid
awsordertypeid
awsacquisitionrecommendationid
applicationid
inventionid
grantedpatentid
cooperationpartnertransfer
id
name
ispositive
active
awsordertype
id
name
isorder
active
valuationdeadlinedays
awsacquisitionrecommendation
id
name
ispositive
active
invoice
id
applicationid
grantedpatentid
entryinvoiceon
contractor
service
invoicenumber
amount
amountjku
amountjkupaidon
invoicepassed
passedamount
passedto
passedon
passedamountreceivedon
inventionid
annualfeedoneuntil
isdeleted

json.log:

Each request entering the authentication filter is logged with the information “in”. If the server processes the request without error there is a second line with the information “out”. The following information is logged:

- Date and Time of the log entry
- Unique request identification (can be used to track all log entries of the specific request)
- User id (the database id of the user doing the request)
- Direction (in, out)
- URI-Path
- HTTP-method
- JSON payload (if send with the request)
- HTTP-response code (only in the out direction)
- Handling Time (between in and out, only in the out direction)

log4app.log:

This log file contains information about all other loggers (the json and the userRepo logger are excluded). The following information is logged:

- Date and Time of the log entry
- Type of the log message (e.g. info, error)
- Name of the class the log message was issued from
- Unique request identification (can be used to track all log entries of the specific request)
- User id (the database id of the user doing the request)
- Information logged from the underlying logger

userRepo.log:

This log file contains information from the user repository. The user repository is used to maintain a copy of the user data within the application memory. In that way it is not necessary to get the user information from the database each time the information is needed.

The following information is logged:

- Date and Time of the log entry
- Type of the log message (e.g. info, error)

- Name of the class the log message was issued from
- Unique request identification (can be used to track all log entries of the specific request)
- User id (the database id of the user doing the request)
- Information logged from the underlying logger

Curriculum Vitae

Personal Information

Günther Haffner, BSc PMBA

Breinbauerweg 8
4040 Linz / Austria
E-Mail: g@haffner.co.at
Date of Birth: 19.09.1974

Education

2014 – present	Master in Computer Science – Software Engineering Johannes Kepler University, Linz, Austria
2014 – present	Master in Business Informatics Johannes Kepler University, Linz, Austria
2012 – 2014	MBA for Financial Management Johannes Kepler University, Linz, Austria
2013 – 2014	Bachelor in Business Informatics Johannes Kepler University, Linz, Austria
2009	International certification for Project Management Level C Project Management Austria, Vienna, Austria
2001 – 2002	Certification as CRM Project Manager Danube University Krems, Austria
1995 – 2000	Business Informatics (not finished) Johannes Kepler University, Linz, Austria
1989 – 1994	Leaving Certificate Commercial Highschool I, Wels, Austria

SWORN DECLARATION

I hereby declare under oath that the submitted Master’s Thesis has been written solely by me without any third-party assistance, information other than provided sources or aids have not been used and those used have been fully documented. Sources for literal, paraphrased and cited quotes have been accurately credited.

The submitted document here present is identical to the electronically submitted text document.

Linz, Date

Signature