Project ID: MXC-0550

INTERSTAATLICHE HOCHSCHULE FÜR TECHNIK BUCHS
HEAT PUMP DATABASE AND SEARCH TOOL IMPLEMENTATION

A Major Qualifying Project Report
Submitted to the faculty of the
WORCESTER POLYTECHNIC INSTITUTE
In partial fulfillment of the requirements for the
Degree of Bachelor of Science

AUTHORS

Kyle Carrero
Matthew Piazza
Kyle Sposato

FACULTY ADVISOR

Michael J. Ciaraldi
Senior Instructor, Department of Computer Science
Worcester Polytechnic Institute

SPONSOR

Stefan Bertsch, PhD
Head of the Institute for Energy Systems
Interstaatliche Hochschule für Technik Buchs

Abstract

This Major Qualifying Project is the creation and deployment of a database management system and web application for the Institute of Energy Systems (IES) at the Interstaatliche Hochschule für Technik Buchs (NTB). This database stores the test results and conditions from the Wärmepumpen-Testzentrum (WPZ), the Heat Pump Test Center. This web application provides a user-friendly heat pump search and comparison tool for use by homeowners as well as a graphical database editing tool that allows WPZ technicians to edit and add new test data to the database with ease. The web application and database were demonstrated by performing user tests with NTB employees and deploying the web application and database on the NTB website.

Acknowledgements

We would like to acknowledge Worcester Polytechnic Institute for providing us with the opportunity to complete our MQP at the Switzerland Project Center. We would also like to thank our advisors Michael Ciaraldi and Dr. Daniel DiMassa, and the Director of the Switzerland Project Center, Dr. Nancy Burnham, for their guidance throughout the preparation and completion of this MQP.

We would also like to thank our sponsors at the NTB Interstate University of Applied Sciences of Technology Buchs for providing us the opportunity to work with them and for their support throughout the project. Specifically, we want to thank Dr. Stefan Bertsch, Dr. Emine Cagin Bertsch, Mick Eschmann, and Markus Markstaler for their assistance in guiding our group.

Executive Summary

Our sponsor, the Wärmepumpen Testzentrum (WPZ), a holder of an ILAC MRA Accreditation and a Swiss Accreditation, is a member of the Institute of Energy Systems within the Interstaatliche Hochschule für Technik Buchs (NTB) and is responsible for the testing and certification of commercial heat pumps against a variety of testing standards developed by governmental and scientific regulatory organizations. Up until this project, the WPZ had been using Microsoft Excel files to store and display the results from various tests they performed on heat pumps. However, they considered this method of data storage inefficient and anticipated difficulties maintaining it in the future. It also hindered readability due to the overwhelming amount of data displayed at once and the technical knowledge required to understand it. In order to address this issue, we created an online database with a more intuitive interface for collecting and storing data, and we connected it to a web application by which homeowners may find heat pumps suited to their needs.

A previous graduate student, Fabian Lutz, developed a project known as the Data Warehouse which acted as a precursor to our database. Though written in software that was incompatible with our systems and which required some restructuring to suit our needs, his project gave us a foundation on which to build our database. Acknowledging that regulations and testing standards would continue to evolve, we designed a schema flexible enough to handle new testing information without having to edit the existing table structure. We also expanded and generalized the database from handling a single heat pump type to handling the various types tested by the WPZ.

Along with moving the primary storage location of the WPZ's data online, we also provided a feature to export and download the database as Microsoft Excel spreadsheets for those still familiar with the old format and for sharing of the test results en masse. With no guarantee of further maintenance, this feature needed to be flexible in parsing and exporting the database in order to handle future changes to the test result data as might occur through changes in the regulations and testing standards. Our database design enabled us to create a reliable, dynamic system for exporting that could extend itself through reading the database tables, whose values defined the structure and fields of the test results.

Finally, we created a website to host both a tool for searching and comparing the various heat pumps and a tool for database maintenance and addition of new test results. The website was designed to be responsive and viewable regardless of device or screen resolution. Due to complications with hosting our site on the NTB server, we utilized an external third-party server and pointed to it with DNS settings from within a subdomain of the NTB website.

We designed our search-and-comparison tool for homeowners looking to install a new pump within their homes. This tool allows homeowners to search for heat pumps whether they know the exact specifications required for their home or only the yearly fuel consumption of an existing heat pump from which they were upgrading. The maintenance section of the website provided WPZ technicians a simple and user-friendly tool to edit the otherwise complicated back-end database through a secure login. We also implemented an interface specifically to ease the process of adding new test results to the database and validate the new data.

In creating the website and database for the WPZ, we addressed the concerns they raised of data storage efficiency and future maintenance, and we also provided a tool making their database more readable and useful to public users. The maintenance tool reduces the risk of data redundancy and relieves the possibility of concurrency and synchronization errors between machines. We actively avoided introducing significant dangers, such as SQL injection or lack of authentication, and designed the system to ensure backwards-compatibility through the export process as well as compatibility with known possible changes to the test result process in the future. As of project completion, the website was deployed onto the following sub-domain: “http://wpz.energiewerkbank.ch/”.

While we successfully created a system supporting the testing results of heat pumps, our database was not designed to account for the hot water boilers that are also tested by the WPZ, and introducing such compatibility will require significant changes to the schema. Our current user authentication system provides a measure of security for the administration portions of the website, but integrating the NTB authentication system would provide NTB’s IT department greater control over the access of those restricted portions. We recommend future efforts be focused towards redesigning the database schema to be able to store test results for hot water boilers, creating a corresponding search tool, and integrating the website and database into the NTB infrastructure.

Table of Contents

Chapter 1: Introduction

The Wärmepumpen-Testzentrum (WPZ) department of the Interstaatliche Hochschule für Technik Buchs (NTB) university has been utilizing multiple Microsoft Excel files to store data acquired from various tests performed on different heat pumps. This method of complex data storage is considered inefficient and difficult to maintain by the NTB Institute of Energy Systems (IES). Furthermore, this data storage method complicates distribution and interpretation of the test data. The Microsoft Excel spreadsheets (converted to PDF files) are distributed via the NTB website, and visitors must then interpret them in order to find which heat pumps suit their requirements. These spreadsheets are considered difficult to understand for the target audience, homeowners and industry members, because of the background technical knowledge required in order to decipher the data. Additionally, the format of the spreadsheets is complicated and presents the user with an overwhelming amount of data that may not be relevant to their requirements.

In order to address the problematic data storage method for the NTB WPZ, we created a database to more effectively store and maintain the heat pump test data. By addressing the WPZ data storage problem through the implementation of a database, NTB is now able to easily store and maintain their heat pump test results with a higher efficiency than that of the previous Microsoft Excel spreadsheets.

To make the data easier to interpret for homeowners, we created a web application that provides multiple methods by which to view the test data in a manner that grants an improved user experience while interacting with the NTB website. As of project completion, the website was deployed onto the following sub-domain: “http://wpz.energiewerkbank.ch/”. This application allows homeowners to input either some data that an installer would provide them or some information on their current heat pump, and it subsequently presents them with a list of suggested heat pumps that fit their needs. Given the list of suggested heat pumps, a homeowner can then select their top choices to compare in more detail in order to make a more informed decision about the purchase they plan on making. This application will greatly benefit homeowners by providing a more user-friendly method of finding heat pumps that match their needs.

In addition to the search and comparison tool, the web application provides a graphical interface to manage the contents of the database. This allows WPZ technicians to edit, add, or remove test data from the database. This feature allows them to retain direct control over the data while having a centralized data storage system. This also reduces data redundancy and improves data consistency and integration with software and web applications.

In the following sections, we will introduce some background information about NTB and the WPZ, after which we will discuss our design and its implementation. Finally, we will summarize our results and identify areas that would benefit from continued development in future projects.

Chapter 2: Background

The IES is composed of approximately thirty researchers and engineers spanning three areas of research: Thermal Energy Systems (TES), Electrical Energy Systems (EES), and the Wärmepumpen Testzentrum. The main focus of the IES is to perform research for small- and medium-sized enterprises. In addition to their research activities, there are three professors and one lecturer who teach system engineering and the Master's degree course, MAS Energy Systems, at NTB. Those within the TES area of research focus on the development of heat pump and refrigeration machines, the design of thermal systems, and the simulation of thermal and fluid systems. The EES's research focuses on the design and development of power electronics and circuit design. The WPZ, our sponsor and primary connection within the IES, is responsible for the testing and certification of commercial heat pumps.

The goal of the WPZ is to test heat pumps against a variety of testing standards that are developed by government and scientific regulatory organizations. These standards and regulations must be met before heat pump manufacturers can enter their pumps into market, and these manufacturers utilize the WPZ in order to certify that their products adhere to all required standards. With the ILAC MRA Accreditation and Swiss Accreditation that the WPZ holds, clients can trust that the obtained test data is reliable. In addition to certifying heat pumps, the WPZ test results provide a basis upon which research can be conducted in order to improve upon current heat pump technology. During their tests, the WPZ investigates various heat pump characteristics such as performance, sound levels, heating and cooling capacity, refrigerant types, and more.

After a manufacturer's heat pump is tested by the WPZ, the test results are released to the public with the manufacturer's consent. As a result, the WPZ is able to provide homeowners, business owners, and other customers with data that can assist them in selecting a heat pump that best fits their requirements. This can significantly help those who are looking to find a reliable system which meets specific constraints.

2.1 Issues with Data Storage

Previously, the WPZ stored their testing data within Microsoft Excel spreadsheets. While this storage method had met their needs, they anticipated difficulties in continuing to keep track

of their ever-expanding data. In addition, working off of spreadsheets posed the concern of concurrent updates. If two technicians were to each begin editing their own copy of the master spreadsheet, it would be unclear which of their new copies would become the master. This would also mean that one technician would have to re-enter their data once the master copy is determined. Duplicated data is also a concern when sharing these spreadsheets across a team. In addition, there is no central data storage, nor is there much of a backup system other than making copies of the spreadsheet, which can easily be lost in an unorganized workspace.

In using spreadsheets to store their testing data, the WPZ also limited their ability to share said data with homeowners whom this information would help. The WPZ website previously just hosted copies of the spreadsheets for end users to download and read. The issue with this was that the spreadsheets are very technical in nature, and only users who had previous knowledge with the testing information could fully understand what it all meant. This made it more difficult for end users to be informed about the varying performance between the heat pumps they were potentially going to purchase.

For these reasons, the WPZ requested that we develop a database and search tool that could be implemented into their website and would solve these issues. Database systems provide a central storage solution, preventing the issue of data redundancy. They can also handle multiple commands at a time, preventing the issue of concurrent updates. Some database systems also allow for automatic backup of the data stored within, protecting their data in case of failure. It is also very easy to integrate a database system into website, therefore providing a greater amount of control over the data and how it is presented to the end user. This means that the homeowners looking for information on heat pumps can be presented with the data in a more easily understood and user-friendly format.

Chapter 3: Methodology

3.1 Design

In order to accomplish our end goal of creating a web application for homeowners to search for the heat pump that best suits their needs, we developed three objectives. First, we would create a database for the test results after exploring the advantages and disadvantages of different possible approaches. Next, we would implement a feature to export our database to Microsoft Excel spreadsheets for simple viewing and sharing. Finally, we would upload our web application to a server after exploring various hosting options.

3.1.1 Database Considerations

In order to plan out the structure of the database, we first needed to understand the current structure of the data and how each entry in the Microsoft Excel spreadsheets related to one another. After conferring with Dr. Stefan Bertsch, he explained to us the testing process for the heat pumps that the WPZ uses as well as what specific data is collected. We discussed with Dr. Bertsch the different use cases for the program we would be writing, and together we analyzed the data that was currently being collected and what was and was not necessary for our application moving forward. These use cases included information such as what types of searches should be available (e.g. the installer information search and fuel consumption search, discussed later), what data needed to be provided by the user for each type of search, and the workflow that the user should follow when searching for heat pumps.

A student working under Dr. Bertsch, Fabian Lutz, completed a similar project only days before we arrived. He created a relational database and import tool for the data; however, it did not meet all the needs the WPZ had in regards to this database and search tool. We were given permission by Dr. Bertsch to use what we could from his report. Mr. Lutz had already created the necessary schema and import tool, but it was unlikely that the IT department at NTB would be able to install and prepare the necessary components of his project in time for us to utilize them in our work. In addition, Mr. Lutz had used Microsoft SQL Server, a database software designed exclusively for Microsoft Windows. As such, we decided to build our own database and import tool using Mr. Lutz's work as a template. We came to the decision that we would edit his

existing database schema, add and remove fields as necessary, and finally translate that schema to a database system that would be more agnostic of operating systems.

We looked into several different database management systems to see which would work better for this project. A comparison of these database management systems can be found in *Appendix A, Table 1*. We decided to translate the schema to work with MySQL (MySQL AB, 1995), a relational database management system that was already in use at NTB. There were several members of the WPZ staff who were very familiar with MySQL, so we were certain that the database could be maintained after this project came to an end. In addition, MySQL also runs on a plethora of operating systems and is free to use.

The import tool, however, would be not be used in our final product. The Microsoft Excel sheets that were previously used by the WPZ contained slight differences that would make creating a generic import tool very difficult to achieve. After analyzing these spreadsheets, we came to the conclusion that in creating an import tool, we would have to set a rigid set of guidelines that would have to be adhered to by all of the technicians in the WPZ. This would make the spreadsheets the "master" copy of the data, rather than the database, and would not solve the current issue that they were facing. This would merely make the database a backup in case they ever lost the "master" copy of the data, which was not its intended purpose. In addition, we felt that this would eventually lead to a scenario where the regulations and testing standards used would evolve in such a way that would require the set of guidelines for the import tool to be altered. These rigid guidelines would not allow for such edits without the import tool needing to be heavily edited and tested to conform to this new structure.

Instead, we decided that the technicians should insert their test data directly into the database, removing the need for an import tool altogether. We would build this functionality into the website through a database maintenance tool, hidden behind a user authentication service. The WPZ technicians could enter the same information into the database through this tool as they would have previously entered into the Microsoft Excel spreadsheets.

3.1.2 Exporting Data

Alongside creating a web application to display the heat pumps online, we also created a tool to export the test results from the MySQL database to Microsoft Excel spreadsheets for sharing, analysis, and simulation purposes. Spreadsheets are the most familiar and compact

format for the technicians to view the results in, but to achieve an easily comprehensible format requires processing and conversion to synthesize all the data from the various database tables together. In preparing to export the test result data, we explored a number of possible approaches, separated primarily by compatibility with the system through which we would develop the web application.

As we began our research we explored options within JavaScript (Eich, 1995) and jQuery (The jQuery Foundation, 2006) due to their natural role in web development and ease of integration into a website. Within the available jQuery libraries, however, we did not find modules for this specific type of data transformation. While JavaScript also did not contain modules for this complete transformation from SQL to Microsoft Excel, it did contain a variety of similar libraries for each half of the conversion; namely, from SQL to JavaScript arrays and from JavaScript arrays to Microsoft Excel. Since we found that JavaScript facilitated the conversion most efficiently when writing to CSV files, and because we were given freedom to modify the layout of the spreadsheets, we explored the possibility of exporting using CSV. However, we decided against it primarily because it would prevent styling the spreadsheets, which we desired to do in order to maintain continuity with the appearance of the sheets previously used by the technicians.

Following research on JavaScript and jQuery, we turned to Python (Python Software Foundation, 2001) with its multitude of advanced plugins, object-oriented classes, and minimal syntactic overhead, using the working assumption during this design phase that even though Python is not native to all systems, it and any requisite modules would be able to be installed on the server we chose to host our web application on. After comparing file format compatibilities of various modules along with intuitiveness and unique features of the interfaces, we settled on the Pandas module (PyData, 2016) as our primary data management system within Python. A comparison of module compatibilities with varying file formats may be found in *Appendix A, Table 2*. Pandas offered the ability to export standard Microsoft Excel sheets instead of exclusively the CSV format that many of the other modules we researched were restricted to, and it also provided a simple and concise interface capable of handling both sides of the data conversion between MySQL and Microsoft Excel files.

In addition to its capabilities in the area of converting, the Pandas module provided support for styling Microsoft Excel files while writing data to them. Though the styling feature

of the Pandas module itself was experimental and did not provide us the full suite of necessary options, the module also provided wrappers to integrate smoothly with the XlsxWriter module (McNamara, 2013), which offered broader functionality and all the styling features we required. This guaranteed that as we exported our data to the spreadsheets, we would be able to maintain a similar style to the formatting system familiar to the WPZ technicians.

During the design process we encountered an issue in deciding how to read and store the data in a conversion layer within Python before writing it to the spreadsheets. It would be simple to read the tables from the database and just write each one to its own sheet in a Microsoft Excel workbook, but due to the desired spreadsheet format of grouping the information by the heat pump which the data referred to, this would not satisfy the objectives of intuitively displaying the test results. Despite the encapsulation and interfaces provided by the Pandas module, mapping the table data into the format for the spreadsheets would not be possible without the aid of a medium in which to store the parsed data. Knowing that there were different forms of test procedures and results for each of the various types of heat pumps, we designed a class structure containing an abstract class for the heat pumps and concrete classes for each of the three different variations. The class diagram is shown in *Appendix B, Figure 1*. Beyond defining some common attributes and methods for the concrete classes to use within themselves, the abstract class also provided a place for methods that were used by the top-level executing script to handle a collection of heat pumps regardless of their specific type. The primary goal in this design was to promote a consistent structure for the data and consistent sharing of common functionality across the various heat pumps, easing the processing and formatting of the data.

3.1.3 Website

3.1.3.1 Hosting

During our investigation of the NTB website and IT infrastructure, several issues were discovered that introduced the possibility for the web application and database to be hosted by a third party. As a result of the encountered complications, the website and database were decided to be hosted by a third-party company.

There were two main reasons that led to utilizing an external host. First, the NTB website used a content management system called Typo3 (Skårhøj, 2005) that constrained its ability to

utilize the technologies necessary to implement our design for the database and website. The second concern was that we would have to request many back-end modifications to the NTB website in order to deploy the database and application. This process was considered to be time-consuming and would have significantly delayed the development of the database and web application. Lastly, when our team arrived to NTB, our IT contact, who manages the website, was noted to be absent from the NTB campus for several weeks. Therefore, we were unable to determine the details required to host the database and web application on the NTB IT infrastructure.

To determine which external website hosting platform best suited the project, we created a list of hosting requirements.. The requirement list that we developed included: hosted in Switzerland, includes a domain, PHP, Python, MySQL, Shell access, FTP/SSH connections, and automated backups. We researched seven third-party hosts in total, which are listed along with their available features in *Appendix A, Table 3*.

Based on *Appendix A, Table 3*, DjangoEurope was considered to be the best external host for this project. The features deemed most vital, which this host provides, are shell access, the ability to use the Python language, and a MySQL database. After conferring with Dr. Stefan Bertsch, we were informed that our maximum budget for any services that must be used by the university for this project is Fr. 500/year. DjangoEurope (DjangoEurope, 2009) hosting was estimated to cost approximately Fr. 90/year. A dedicated IP address or domain name was not required for this project. A canonical name record was added to the NTB DNS settings to point to the third-party host's server. DjangoEurope guarantees that the IP address of the server does not change. In total, hosting the project on a third-party server costs approximately Fr. 90/year.

3.1.3.2 User Experience

This section describes how users were expected to interact with the front-end designs. Additionally, this section provides a general flow of events for each major feature on the website.

Heat Pump Search Methods *(Appendix D, Figure 1)*

In order to search for heat pumps, the desired heat pump type must be selected. Afterwards, two separate methods were developed to search for heat pumps. The first method,

the “Installer” search, would be used when the user has been provided information (i.e. heat pump type, power consumption, outside temperature, and the building’s heating system) by an installer. The second method, the “Fuel Consumption” search, would be utilized by users who already know what power output is needed in addition to how much fuel is supplied to the heat pump.

Heat Pump Comparison *(Appendix D, Figures 2, 3, 4)*

The heat pump comparison page was created to display attributes of heat pumps that were selected on the search results page. This feature was designed to compare a limited number of heat pumps whose identifiers are maintained in a “comparison list”. After users select heat pumps to compare, the heat pump identifier is stored, and a separate “comparison” page lists the attributes of the heat pumps that appear in the “comparison list”. In order to change the selected heat pumps, users must either start a new search or navigate back to the search results page. When navigating back to the search results page, the previously selected heat pump checkboxes are already checked. When unchecked, the heat pump is removed from the “comparison list”. Executing a new search clears the entirety of the “comparison list”. Lastly, each heat pump’s full test results can be viewed from the search results page by clicking on a heat pump’s test number.

Database Management *(Appendix D, Figure 5)*

The database management section of the website allows WPZ technicians to modify the back-end database. This section is accessible through a user login which requests a username and password. Tables in the database that would need to be modified in the future would become accessible upon logging in and can be manipulated with three basic operations: create, update, and delete.

Test Result Input *(Appendix D, Figure 6)*

The test result input page was designed to provide WPZ Technicians a simple means of adding new test results to the database. This section is accessible through a user login that requests a username and password. The form was designed to capture all of the data necessary to create a new test result and verify that the data is correct. User feedback was important in this process considering the form is dynamic and consists of over twenty inputs. The form was

designed to provide feedback to the user if an input was left empty or invalid. Additionally, the server was configured to give feedback based on the success or failure of inserting the test result data.

3.1.3.3 User Interfaces

This section describes the front-end designs of major features of the website. In general, the website was designed to be responsive. This allows the website to be viewed on any device or screen resolution. Responsivity was achieved through the use of Cascading Style Sheets (CSS) media queries and Bootstrap's (Bootstrap, 2011) mobile-first design. Media queries provided the ability to apply specific styles based on the users device type and screen resolution. Lastly, form validation was an important aspect of the user interfaces throughout the website. If the server detected invalid form input, a proper message was returned and displayed to the user in a red box with an appropriate error message.

Heat Pump Search Methods

The two search methods were designed to be accessed through simple forms contained in an "Accordion" dropdown on the index page. The "Installer" search method requires the following inputs: heat pump type, power consumption, outside temperature, and type of building heating system. The "Fuel Consumption" search method required the following inputs: heat pump type, fuel type, fuel consumption, building heating system, and whether or not a hot water system already exists. The inputs required to complete these searches were designed to be entered in the form of text boxes, selection dropdowns, and radio buttons.

Heat Pump Comparison

After users executed a search and received results, each heat pump listing was designed to display data pertinent to each heat pump such as manufacturer, model, heating type, and heating power. Additionally, a checkbox at the bottom of each heat pump listing was used to add heat pumps to, or remove from, the comparison list. A separate page was created for users to view and compare attributes of the selected heat pumps. On this page, a table was designed to list the selected heat pumps, where each row was an attribute of the heat pumps such as sound level, refrigerant type, test number, and power consumption.

Database Management

The database management interface (DBM) was designed to be a simple and user-friendly tool to edit the otherwise complicated back-end database. The DBM design consists of a list of all the tables in the database. From this interface, WPZ technicians can select which table they wish to modify which redirects to a page that displays the contents of the selected table. Each row of the displayed table has buttons for deleting and editing. Additionally, there is an "Insert" button which adds new rows to the table. Both the "Insert" and "Edit" buttons utilize a modal form that display the necessary inputs.

Test Result Input

The test result input interface was designed as a form with a variety of input types. Additionally, the form had to be dynamic, considering one test result could have multiple test conditions. This form was designed to be several groups of text boxes, selection dropdowns, and a single radio button to contain all of the information necessary to insert a new test result into the database. In order to account for the case where there were multiple test conditions, buttons were added to enable the addition and deletion of additional test condition inputs.

3.2 Implementation

With the design phase completed, we set out to implement our designs. We began by altering and improving the database schema provided to us by Fabian Lutz. After transferring the WPZ's data into the database, we set out to accurately export the data as Microsoft Excel spreadsheets in the same format as they were given to us. Finally, we created a website to host the search and comparison tools, as well as tools for insertion of new test data into and maintenance of the database, which became the WPZ's primary repository for both new and previous heat pump test results. The source code for the database, website, and export tool is located on Github at the following URL: "https://github.com/SwitzerlandMQP-A17/WPZ_Website_and_DB".

3.2.1 Database Iterations

As discussed earlier, much of the work in designing the schema was completed by Fabian Lutz prior to our arrival. However, it was written in Microsoft SQL Server syntax, and therefore had to be changed to be more operating system agnostic. The conversion between Microsoft SQL Server and MySQL syntax was fairly trivial, only requiring the replacing of some keywords. There were, however, several quirks that we had to address in his implementation. Mr. Lutz had only designed his database around a single type of heat pump, the "Air/Water", or "Luftwasser", style of heat pump. This meant that when trying to input data from the other two styles of heat pump, there would occasionally be some issues. Due to evolving regulations and testing standards, we also needed to implement a system which would be flexible enough to handle new testing information without having to edit the existing table structure. To remedy this, we edited his original design to be more agnostic of specific heat pump types, and we attempted to make the schema as flexible as possible around the types of data that were expected to change, as shown in *Appendix C*.

To make the schema more pump-agnostic, as well as more easily understood, we altered the names and types of certain fields in order to make it more easily understood what each table was for. For example, the Lufttemperatur field, meaning "Air Temperature", in the Bedingung table was changed to be Umgebungstemperatur, meaning "Ambient Temperature", which was seen to be a more agnostic name for that field, since not all pumps used air. The Fakten table, meaning "Fact", was also changed to be named Verbindung, meaning "Link" or "Connection", to reflect its purpose as a linking table. We also removed several unnecessary columns from the schema, such as the Norm_Typ column from the Bedingung table, the Resultat_Part2_ID field from the Fakten table, and the Referenzwert_Low and Referenzwert_Medium columns from the Info table. We felt that removing these unnecessary and unused fields would remove any unnecessary bloat from the database.

In order to make the schema more flexible for future testing standards, we needed to edit how the database linked these standards to the test results. New standards can get added, older standards get updates every few years, and a test can have multiple standards that it is testing for. In Mr. Lutz's design, the testing standards, or Norms, were being stored in a one-to-one relationship with each set of test results. This would not work, as we needed a one-to-many relationship between test results and the testing standards. So to accomplish this one-to-many

relationship, we removed the NormID field from the Fakten table and instead created a NormInfo table that would link together NormID's with any InfoID's that use them. This would allow the set of tests for a given heat pump, unified under a single InfoID, to have any number of testing standards linked to it in this table, thus providing that one-to-many relationship.

Once we finished editing the schema, we imported the rest of the data from the spreadsheets using a piece of software that we wrote to generate the insert statements from the Microsoft Excel spreadsheets containing the test data. However, we felt that this piece of software was not reliable enough to allow the university to use it in production. We also had decided that the entering of any new data would be handled on the website. Therefore, after importing all of the existing data and ensuring that it was correctly imported, we did not continue to work on the import tool.

Finally, with all of the data properly loaded into the database, we created the stored procedures that we would use on the website to access the correct information. We decided to use stored procedures as they were much more similar to how object-oriented or functional programming languages worked. The ability to create a function that could perform multiple SQL queries in the background after calling a single command was preferable to manually constructing multiple insert and select statements when interacting with the database from the website.

3.2.2 Exporting Data to Microsoft Excel

As we began to develop our method of exporting data to Microsoft Excel files from the MySQL database, we used the Python 3 programming language for its many easily-installed modules and minimal overhead involved in object-oriented development. We also used the previously discussed Pandas module to handle both the querying of our database and the exporting and styling of this data to Microsoft Excel spreadsheets.

Connecting to the MySQL database required the installation of both the SQLAlchemy (Bayer, 2016) and PyMySQL (Naoki, 2009) supplemental modules, but executing database queries became trivial upon supplying these helpers. Creating styled Microsoft Excel files also required the integration of another module, XlsxWriter, which provided vital formatting features, such as column spacing or text rotation, along with other useful tools such as image insertion.

The greatest difficulty in exporting the data came in implementing the conversion layer in which to store the heat pump characteristics and test results. Beyond just storing the data, this layer also needed to be flexible in parsing and exporting the data in order to handle future changes to the test result data as might occur through changes in the regulations and testing standards. With the aid of our sponsors in determining the possible categories that these changes could occur within, we wrote a system that dynamically parsed these sections of data by reading the tables which defined the data structure. It then used this information to store the actual test result values in an extendable format that could be iterated through during export reliably, regardless of the possible changes.

This flexibility of parsing and storing the data also created a fortunate side effect. Instead of a single abstract class and three concrete classes, as the conversion layer had originally been designed with, we were able to refactor the implementation of the single "AirWaterPump" class to manage all the varieties of heat pump test results. This occurred because of how dynamically the code read the data and because of the similarities in the generic categories of test result data between the various types of heat pumps. In the end, the same principles and methods from this one class implementation were able to be reapplied with some minor improvements in logic in order to perfectly handle all the required sets of test result data.

Alongside the dynamic storage of the test result data, we also implemented a procedure of exporting the data which would remain reliable in the future as well. The Microsoft Excel workbooks were divided into sheets by the category of heat pump type. Doing this allowed us to group the heat pumps by a static category, keeping the spreadsheets organized in a manner that was subject to less change than if they had been grouped by testing standard, as they had been in the original spreadsheet format. Because not all testing standards produce the same data fields, the columns of each sheet cannot be constant, but by dynamically parsing the test results we were able to create a column for each field and design a flexible format independent of any particular testing standard. However, despite ensuring uniformity among results and compatibility with both future and past test results, this method came with a drawback: one column was displayed for each attribute regardless of whether that data was present in the group of queried heat pumps or not. The sheer number of columns necessary grew such that we were concerned with the ability of a user to view and easily absorb the presented data. However, in the final stage of styling the exported spreadsheets, we polished this flaw away. At the last step of

writing the Microsoft Excel files, our export script evaluated all columns to check if any data was actually present in them, and if a column was empty then it would be hidden. This greatly aided the usability and user experience of our design because while the data storage class would expand as necessary for all possible fields, the actual spreadsheets themselves shrank the collection of visible fields down to only those which were actually helpful to a user.

3.2.3 Website

3.2.3.1 Server Side

For server-side processing, we chose to implement the web-server using the Node.js (Joyent, 2009) JavaScript runtime environment. This software was chosen primarily because Node.js can easily execute server-side Python scripts, which were used for importing to and exporting from the database. Furthermore, Node.js includes the open-source Node Package Manager (NPM) which is "The world's largest software registry" (NPM, 2011). Several modules available in the NPM were used to simplify many parts of the server itself, such as HTTP requests and responses, sessions, templating, MySQL database interaction, form validation, logging, and more. The following modules from the NPM were utilized throughout the website: Body-Parser (NodeJS Foundation, 2011), Express (NodeJS Foundation, 2011), Express-Session (NodeJS Foundation, 2011), Express-Validator (Tavan, 2011), Embedded JavaScript Templates (EJS) (Eernisse, 2014), Forever (Robbins, 2010a), MySQL (Wilson, 2011), Path (Jinder, 2014), Python-Shell (Mercier, 2013), and Winston (Robbins, 2010a). A brief explanation of each module and their functionality in the website is provided below.

Body-Parser

This module is middleware software that parses incoming request bodies sent to the server before the response is handled. Given a specific set of options, Body-Parser will take incoming requests and parse their data accordingly. For example, this server utilizes the JavaScript Object Notation (JSON) format to transfer data between the client and the server. When sending JSON data to the server, Body-Parser will intercept the request and parse the data as JSON such that when the data is passed to the route handler, the data is already parsed and easily accessible. Route handlers were used throughout the server to handle specific user requests, such as navigating between pages or submitting a form.

Express

Express is the standard web-framework for Node.js applications that simplifies many aspects of a web-server. While not a server in and of itself, Express is most commonly used to simplify HTTP requests and responses, headers, routing, and templating. Furthermore, Express assists in implementing a server-side Model-View-Controller (MVC) design pattern.

Express was utilized throughout the entirety of the server's implementation. It provides the basis by which client requests are handled by declaring specific route handlers such as *"GET"* and *"POST"*. Additionally, Express provides templating support, which allows the server to render dynamic templates which serve front-end interfaces. The use of templates by the server is further explained in the "EJS" section below.

Express-Session

The Express-Session module was used to store user session data. Express-Session simplifies the process of storing, modifying, and reading session variables.

Session variables are significant to the overall function of the website considering there are multiple user specific variables that must persist on different pages. For example, a session variable was used to store a list of heat pump identifiers that were selected by the user on the "Search Results" page. This list of identifiers was used to populate the "Compare" page, such that only the user-selected heat pumps are displayed. Session variables are also utilized to maintain "logged-in" user information to determine if users are authorized to access certain pages. Lastly, session variables were used to store information such as the name of the table being edited in the Database Management Interface and the currently selected heat pump type for searches.

Express-Validator

The Express-Validator module provides a means of server-side input validation. This module was used to validate user input for both search methods. Express-Validator allows validation procedures to be given to a route signature. A function call to this module executes the given procedures against the data that was passed to the route. This function returns a list of

customizable error messages. This module was useful for providing users with accurate error responses in order to complete a valid search.

EJS

The server's primary task is to send content to the client after it has processed the client's request. In order to provide the client with content, the server "renders" templates using the EJS module. EJS allows JavaScript to be embedded into HTML documents in order to form a template. This allows the server to render pages by passing variables or any other necessary data to the template. These variables can then be accessed in the template and processed by supplementary JavaScript. The output is then embedded into the template as valid HTML. Finally, the completed template, now a valid HTML document, is sent to the user by the server.

Forever

This module is used to indefinitely run the server. Additionally, Forever automatically attempts to restart the server five times should it ever crash.

MySQL

The MySQL NPM module is a driver that allows for interaction with a MySQL database via Node.js. This module provides several features that were significant to the development of the website, such as connection pooling and parameterized queries.

The server utilizes the process of connection pooling in order to execute queries on the MySQL database. This process involves establishing a single "pool" connection to the MySQL database and checking out "clients" which are then used to execute queries. When a client is finished executing its queries, it can be released back to the pool. The benefit of using this process is that connections to the MySQL database do not need to be reopened. Connection pooling eliminates the time necessary to establish a connection to the database and leads to overall lower response times.

This module also incorporates the concept of parameterized queries. These queries are utilized in order to prevent SQL injection attacks. Parameterized queries involve pre-compiled SQL statements with parameters that are inserted at run-time. As a result, user input will always

be used as a parameter of the original query, instead of modifying the original query, which is the goal of SQL injection attacks.

Path

The Path module provides a means of working with files and directories on the server. It allowed the server to establish a base directory such that the use of relative file paths was possible. The Path module is by default available in the NPM registry and does not require any separate installation, unlike the rest of the modules used by the server.

Python-Shell

Python-Shell allows Node.js to reliably interact with Python scripts. This module is used to execute the Python scripts that export the database to Microsoft Excel files (See section 3.2.2).

Winston

The Winston module provides the server with an advanced logging system. Additionally, it simplifies writing to and reading from files. The server uses Winston to log a variety of information from the server into two separate files for errors and general information. Furthermore, Winston is able to catch and log uncaught exceptions. This is significant considering Winston will prevent the server from crashing in the event that an unforeseen exception is generated. Lastly, Winston logs interactions performed by authorized users. This information can be used in order to identify website interactions by users. These logs were saved in the same directory location as the server and could be accessed via File-Transfer-Protocol (FTP) to the DjangoEurope server.

3.2.3.2 Client Side

Bootstrap

Bootstrap is a front-end development framework that provides pre-made CSS, HTML structure, and interactive elements such as modals. Additionally, Bootstrap enforces the concept of mobile-first development such that the front-end is responsive. This ensures that the website can be used on any device at any resolution.

jQuery

jQuery is a JavaScript framework that simplifies event delegation, animation, Document Object Model (DOM) traversal, asynchronous requests, and more JavaScript functionalities. Additionally, this framework is compatible across platforms and browsers. The use of this framework was significant in the website, considering the website makes a multitude of asynchronous requests. For example, in the technician test result input, an asynchronous request is used to attempt to submit the form data. This asynchronous behavior is important, considering the form will maintain the currently entered data even if an error is generated. Additionally, this behavior prevents the form from being cleared when an error occurs, which would require the user to re-enter their data.

I18N

jQuery.I18n (Wikimedia, 2012) is a jQuery plugin that allows for easy internationalization of websites through the use of JSON and its data API. All translations between languages are stored in their appropriate JSON files as key-value pairs. Every language should have the same keys, with differing values depending on the text in that language. Their Data API allows for easy swapping between languages. By adding a “data-i18n” tag to any DOM element, the text within that element can be easily replaced by whatever value goes along with the key specified in the tag.

Chapter 4: Conclusion and Future Work

In creating the website and database for the WPZ, we have successfully remedied all of the concerns that we discussed earlier. Though hosting the website has added a small cost to the WPZ's budget, it has also made their test results more accessible to the public, providing a user-friendly tool that allows homeowners to easily find technical information about heat pumps and determine the one best suited to their needs, without the need of any extensive technical knowledge pertaining to heat pumps. Through transferring the WPZ test results to an online database and creating a tool for its future maintenance and growth, we provided protection against data redundancy, concurrency issues, and loss of data. This transference also improved the WPZ's ability to easily store and maintain their heat pump test results over that of the previous Microsoft Excel spreadsheets, and it enabled future integration with other software and web applications. Though moving to an online server does introduce new risks, we actively avoided introducing significant dangers such as SQL injection or lack of authentication. By including a feature for exporting the database as Microsoft Excel spreadsheets, we preserved the WPZ's previous data storage method's strength of being easily shared and viewed as a whole, and we also provided backwards-compatibility for any systems or users that would need or desire the previous version. We also designed the system to accommodate changes in testing standards and regulations, future-proofing the database and providing a measure of protection against possible changes in heat pump testing requirements.

While we did actively seek to design this system with robustness and comprehensiveness in mind, there are still several limiting factors with our implementation that should be expanded upon for future work. The greatest limiting factor we have discerned is the expandability of our database. While this schema works for the heat pump test results being stored, it does not account for the hot water boilers that are also tested by the WPZ. We were unaware that hot water boilers, while outside the scope of our project, were also tested until we were nearing the completion of the schema implementation. Our current implementation is only designed to store information about heat pumps, and thus, it will need an overhaul in order to be able to store the water boiler test results.

We are currently limited by our user authentication system as well. We implemented a basic user login section to hide the administration portions of the website from the public;

however, this was a temporary fix. Only a single account, an administrator, was created and the password was unencrypted. We were not able to integrate our system with NTB's central authentication system in the time provided for the project; however, we highly recommend that this integration be implemented if possible. Doing so would provide NTB's IT department greater control over who can access those restricted portions of the site, making it much more secure.

We recommend future projects redesign the database schema to be able to store test results for hot water boilers. We also recommend the creation of a corresponding search tool with which homeowners can search for the hot water boiler that best suits their needs. Future projects should also integrate the website and database into the NTB infrastructure to allow IT to more easily maintain this application in the future.

References

Bayer, M. (2011). SQLAlchemy. Retrieved from https://www.sqlalchemy.org/

Bootstrap. (2011). Bootstrap. Retrieved from https://getbootstrap.com/

DjangoEurope. (2009). Django europe. Retrieved from https://djangoeurope.com/

Eernisse, M. (2014). Ejs. Retrieved from http://ejs.co

Eich, B. (1995). JavaScript. Retrieved from https://www.javascript.com/

Jinder. (2014). Path. Retrieved from https://github.com/jinder/path

Joyent. (2009). NodeJS. Retrieved from https://nodejs.org/en/

Lutz, F. (2017). *Aufbau eines data warehouse für wärmenpumpen.* ().

McNamara, J. (2013). XlsxWriter. Retrieved from http://xlsxwriter.readthedocs.io/

Mercier, N. (2013). Python-shell. Retrieved from https://github.com/extrabacon/python-shell

Microsoft. (2017). Microsoft SQL server. Retrieved from https://www.microsoft.com/en-us/sql-server/sql-server-2017

MySQL AB. (1995). MySQL. Retrieved from https://www.mysql.com/

Naoki, I. (2009). PyMySQL. Retrieved from https://github.com/PyMySQL/PyMySQL

NodeJS Foundation. (2011). Express JS. Retrieved from https://expressjs.com/

NPM. (2011). Node package manager. Retrieved from https://www.npmjs.com/

PyData. (2016). Pandas. Retrieved from https://pandas.pydata.org/

Python Software Foundation. (2001). Python. Retrieved from https://www.python.org/

Robbins, C. (2010a). Forever. Retrieved from https://github.com/foreverjs/forever

Robbins, C. (2010b). Winston. Retrieved from https://github.com/winstonjs/winston

Skårhøj, K. (2005). Typo3. Retrieved from https://typo3.org/

Tavan, C. (2011). Express validator. Retrieved from https://github.com/ctavan/express-validator

The jQuery Foundation. (2006). jQuery. Retrieved from http://jquery.com/

Wikimedia. (2012). jQuery.I18N. Retrieved from https://github.com/wikimedia/jquery.i18n

Wilson, D. (2011). MySQL JS. Retrieved from https://github.com/mysqljs/mysql

Appendix A

Database System	Relational	Stored Procedures	Free	Client-Server Model	JSON Support
Oracle					
Microsoft SQL Server					
MySQL					
PostgreSQL					
Sqlite 3					
Key:	Has Feature	Limitations	Doesn't have feature		

Table 1: SQL Database Features

Javascript Modules	SQL	Excel	CSV
json-2-csv			
fast-csv			
sql			
mysql			
sqlite3			
Python Modules	**SQL**	**Excel**	**CSV**
Pandas			
SQLAlchemy			
XlsxWriter			
xlutils			
csv			
Key:	Has Feature	Limitations	Doesn't have feature

Table 2: Data Migration Module Features

HOST	PLAN	PRICE (CHF/mo.)	HOST	DOMAIN	PHP	PYTHON	MYSQL	POSTGRESQL	SHELL	FTP/SSH	BACKUPS
Server Town	Easy	3.8	CH	■	■		■			■	
METAnet	-	6.7	CH	■	■		■			■	■
Cyon	Single	9.9	CH	■	■		■			■	■
HostPoint	Standard	12.9	CH	■	■		■			■	■
Infomaniak	-	14.31	CH	■	■		■			■	■
DjangoEurope	Basic	5.0 (EURO)	EU		■	■	■	■	■	■	■
Amazon AWS	Free	-	EU		■	■	■	■		■	■

Table 3: External Host Features

Appendix B

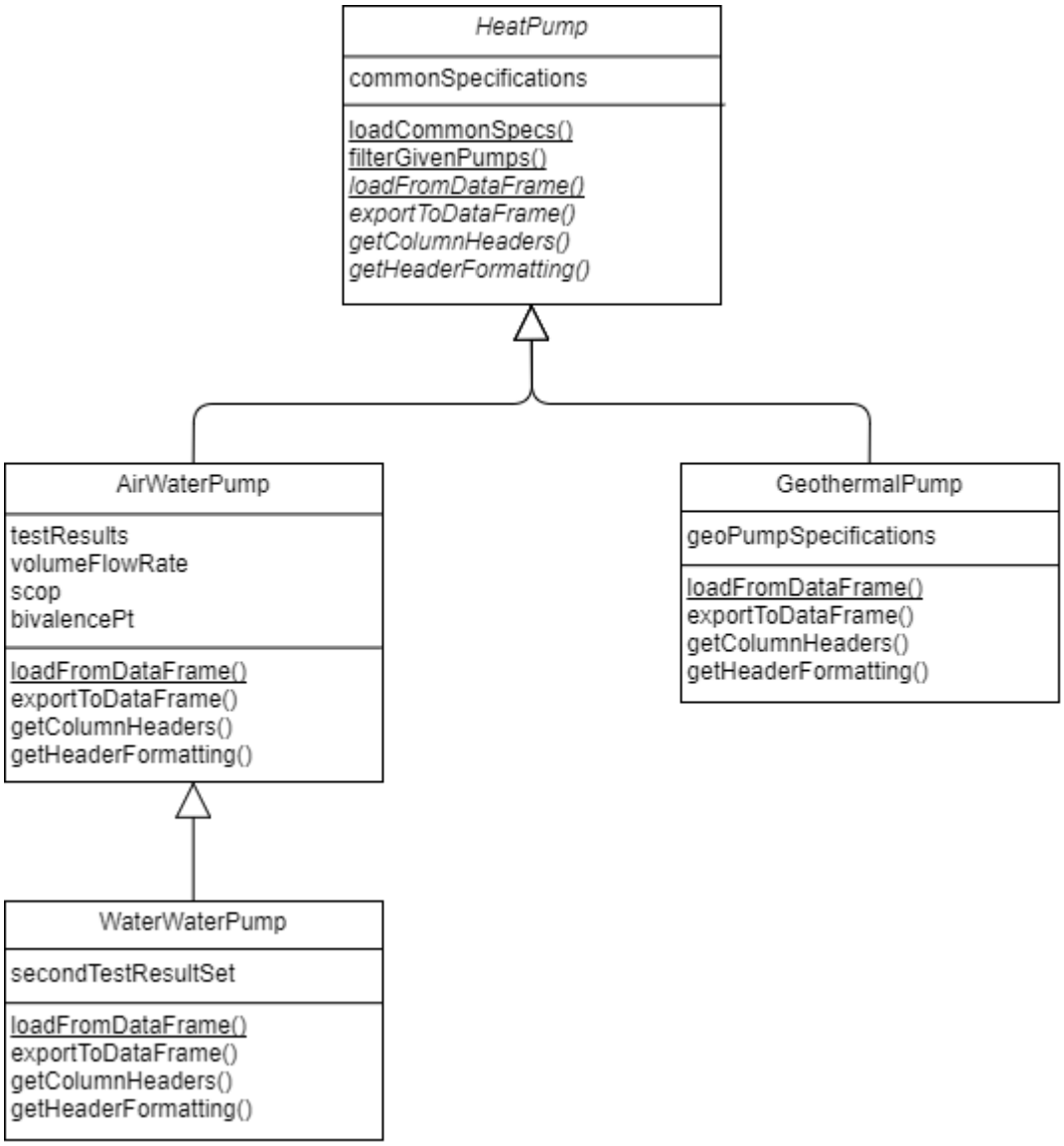

Figure 1: Original Design for Test Result Storage During Export Process

Appendix C

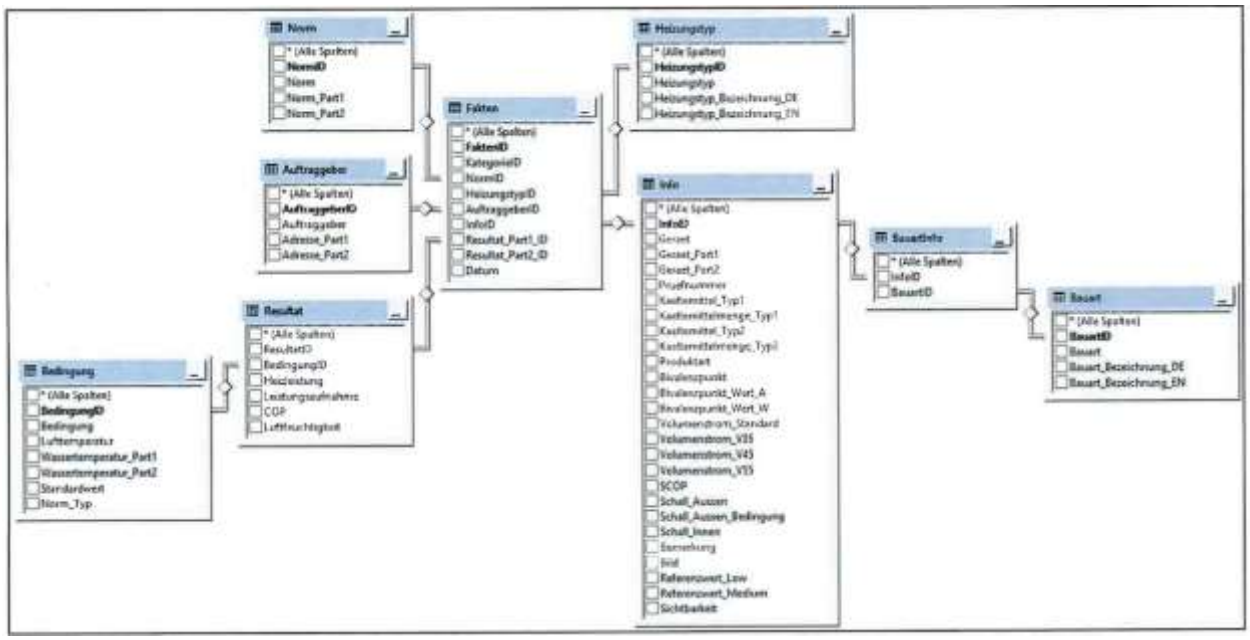

Figure 1: Fabian Lutz's Database Schema

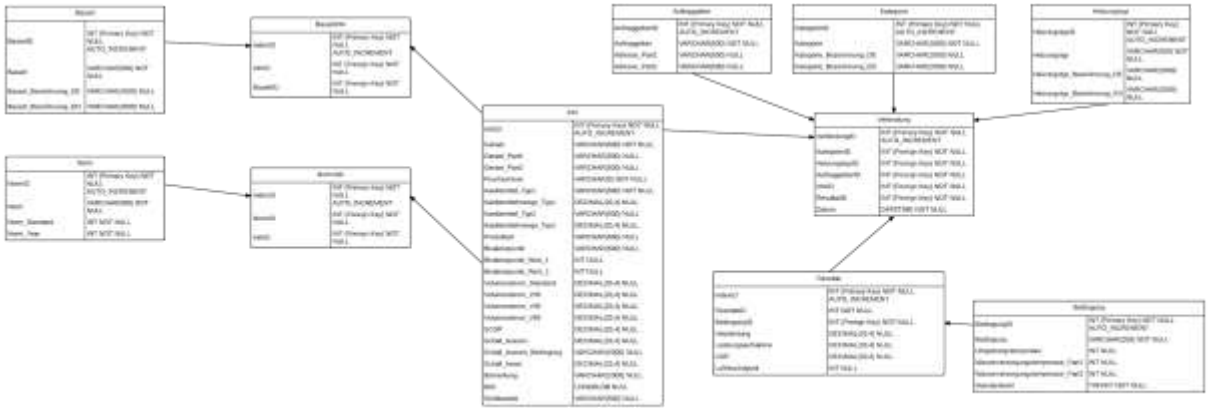

Figure 2: Our Database Schema in German

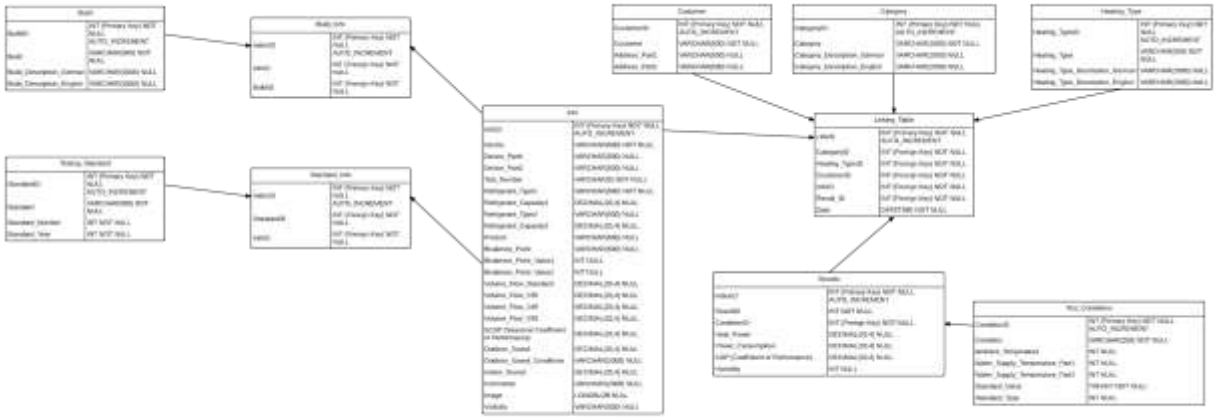

Figure 3: Our Database Schema in English

Appendix D

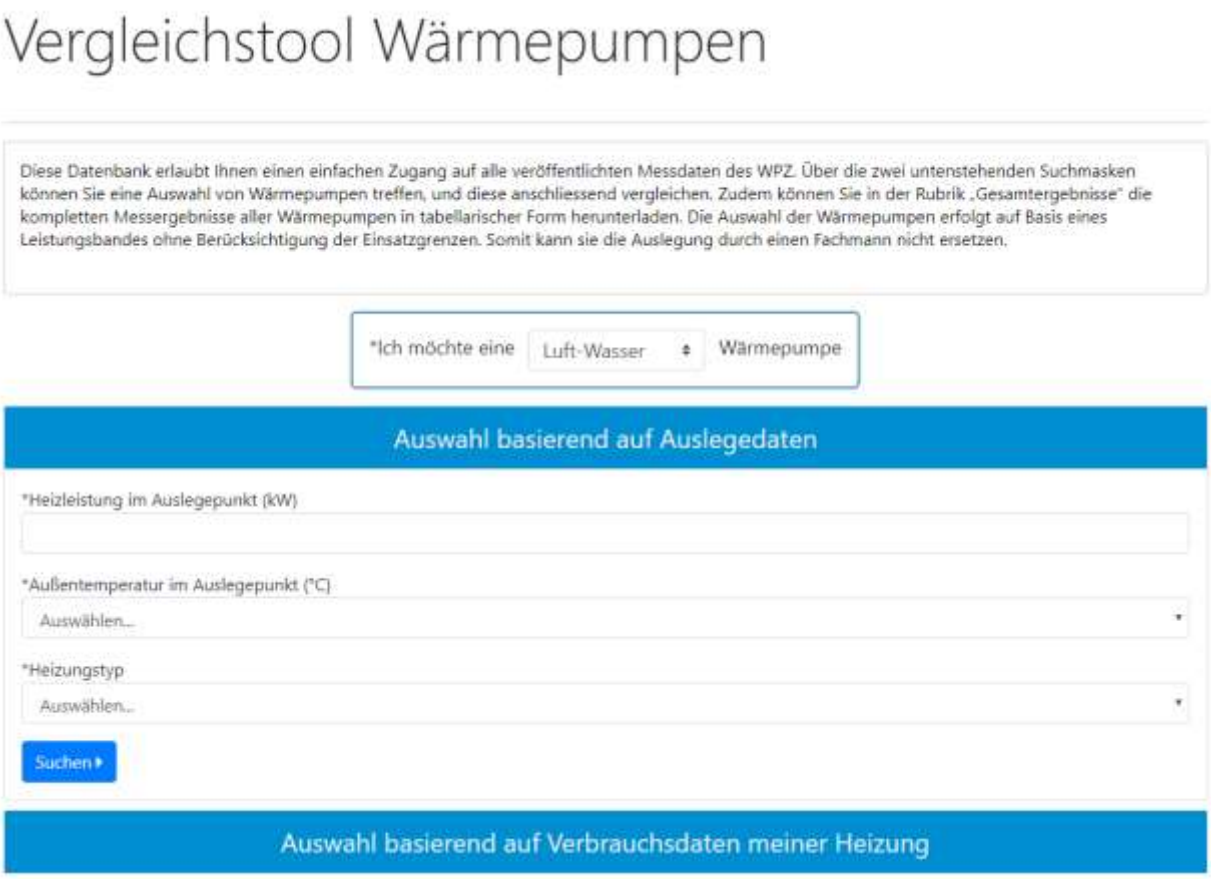

Figure 1: Landing Page and Search Methods

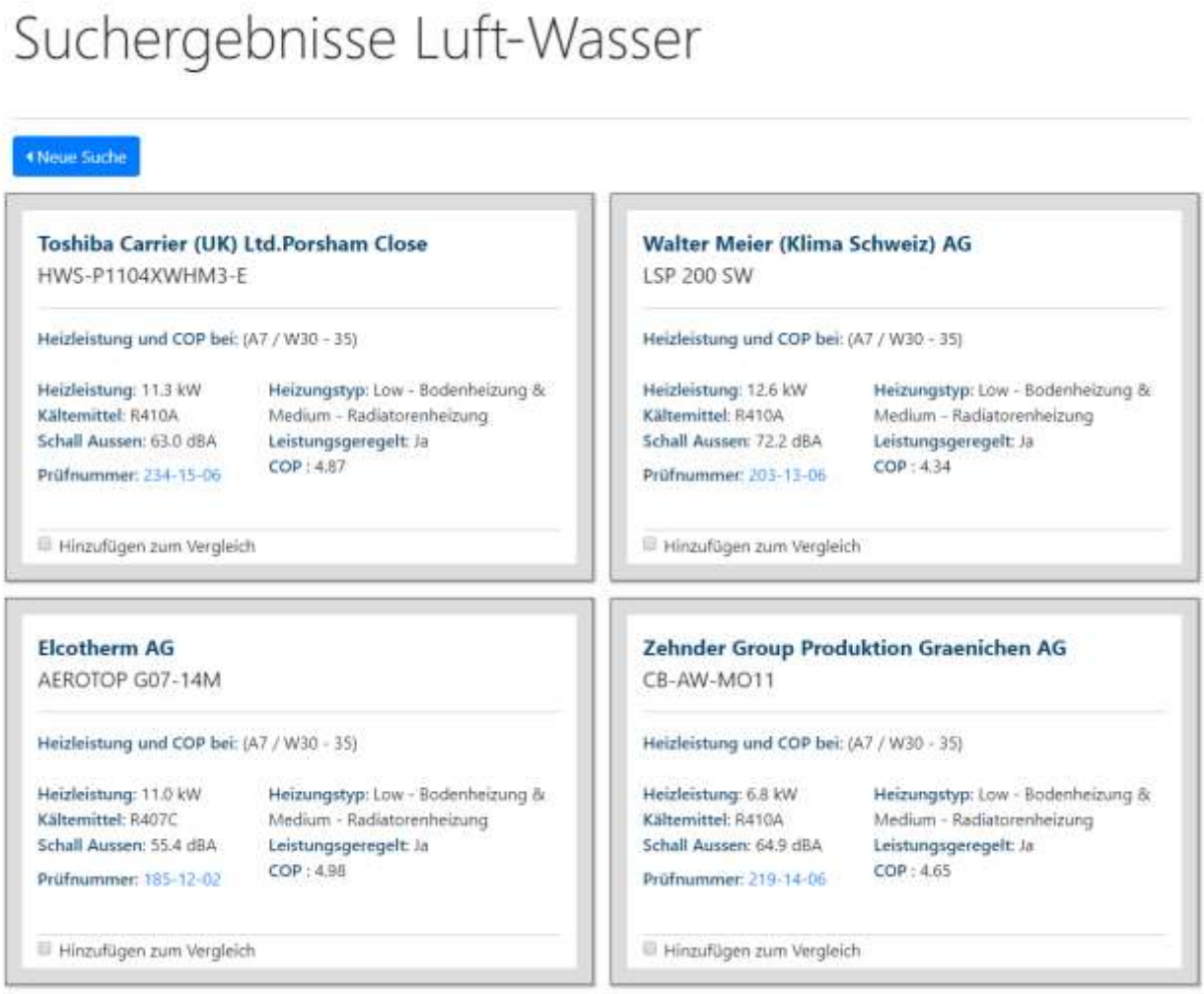

Figure 2: Search Results Listing Four Heat Pumps

Vergleichstool Wärmepumpen

	Toshiba Carrier (UK) Ltd.Porsham Close HWS-P1104XWHM3-E	**Walter Meier (Klima Schweiz) AG** LSP 200 SW	**Elcotherm AG** AEROTOP G07-14M
Heizungstyp	Low - Bodenheizung & Medium - Radiatorenheizung	Low - Bodenheizung & Medium - Radiatorenheizung	Low - Bodenheizung & Medium - Radiatorenheizung
Prüfnummer	234-15-06	203-13-06	185-12-02
Kältemittel #1	R410A	R410A	R407C
Kältemittelfüllmenge #1 (kg)	2.7	5.1	4.6
Schall Aussen (dBA)	63	72.2	55.4
Heizleistung (kW) bei (A7 / W30 - 35)	11.3	12.6	11.0
Leistungsaufnahme (kW) bei (A7 / W30 - 35)	2.3	2.9	2.2
COP bei (A7 / W30 - 35)	4.87	4.34	4.98
Leistungsgeregelt	Ja	Ja	Ja

Figure 4: Comparison Page with Three Heat Pumps Being Compared

Vergleichstool Wärmepumpen Prüfresultate

Toshiba Carrier (UK) Ltd.Porsham Close — **Prüfnummer**: 234-15-06

HWS-P1104XWHM3-E

Kategorie: Luft/Wasser

Bauart: Splitwärmepumpe / Leistungsgeregelte Wärmepumpe mit Frequenzumformer

Heizungstyp: Low - Bodenheizung & Medium - Radiatorenheizung

Leistungsgeregelt: Ja

Normen: EN 14511:2013 / EN 14511:2011

Kältemittel #1: R410A

Kältemittelfüllmenge #1: 2.7 (kg)

Volumenstrom-V35: 2 (m^3/h)

Volumenstrom-V45: 1.85 (m^3/h)

Volumenstrom-V55: 1.37 (m^3/h)

Schall Aussen: 63.0 dB(A)

Schall Aussen Bedingung: A7 / W55

Bedingung	Heizleistung (kW)	Leistungsaufnahme (kW)	COP	Luftfeuchtigkeit (%r.h.)
A10 / W35	18.9	4.1	4.60	80
A7 / W30 - 35	11.3	2.3	4.87	89
A2 / W35	12.1	3.6	3.35	84
A-7 / W35	11.0	4.0	2.77	75
A-15 / W35	10.5	4.2	2.51	
A7 / W40 - 45	10.9	2.9	3.79	89
A7 / W47 - 55	12.4	4.1	3.04	89
A-7 / W55	7.9	4.0	1.96	75

Figure 3: Complete Heat Pump Test Results

Wärmepumpen-Testzentrum Datenbank

AuftraggeberID	Auftraggeber	Adresse_Part1	Adresse_Part2
1	CTC Giersch AG	Bahnhofstrasse 60	CH - 8112 Otelfingen
2	Daikin Europe N.V.	Zandvoordestraat 300	B - 8400 Oostende
3	De Dietrich Thermique	57 rue de la Gare	F - 67580 Mertzwiller
4	Elcotherm AG	Sarganserstrasse 100	CH - 7324 Vilters
5	Energy Panel S.L.	Carretera Estepa-Guadix PK 45	E - 14900 Lucena (Cordoba)

Figure 5: Database Management Interface

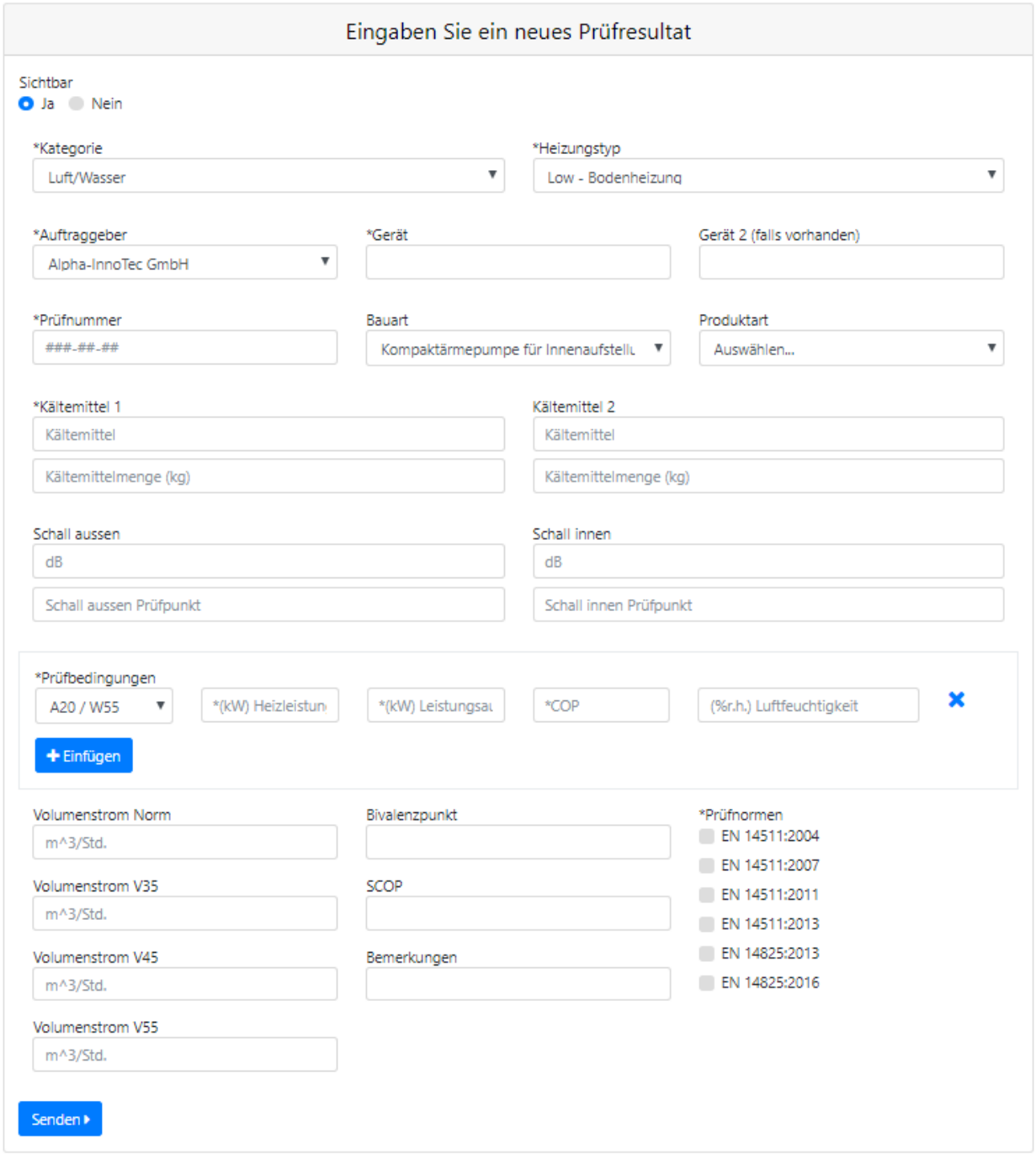

Figure 6: Technician Test Result Input Interface