Database Selection Process in Very Small Enterprises in Software Developmen

A Case Study examining Factors, Methods, and Properties

Teodor Adolfsson
Axel Sundin

Systems Sciences, bachelor's level
2023

Luleå University of Technology
Department of Computer Science, Electrical and Space Engineering

[This page intentionally left blank]

Abstract

This thesis investigates the database model selection process in VSEs, looking into how priorities and needs differ compared to what is proposed by existing theory in the area.

The study was conducted as a case study of a two-person company engaged in developing various applications and performing consulting tasks. Data was collected through two semi-structured interviews. The first interview aimed to understand the company's process for selecting a database model, while the second interview focused on obtaining their perspective on any differences in their selection process compared to the theoretical recommendations and suggested methodology. The purpose was to investigate the important factors involved in the process and explore why and how they deviated from what the theory proposes.

The study concludes that VSEs have different priorities compared to larger enterprises. Factors like transaction amount does not have to be considered much at the scale of a VSE. It is more important to look into the total cost of the database solution, including making sure that the selected technology is sufficiently efficient to use in development and relatively easy to maintain.

Regarding selection methodology it was concluded that the time investment required to decide what is the best available database solution can be better spent elsewhere in the enterprise, and finding a good enough solution to get the wheels of the ground is likely a more profitable aim.

Keywords: Database selection process, Database model, Database management system, Technology selection, Very small entities, Very small enterprises, Software development.

Table of Contents

List of Figures & Tables

Abbreviations

API - Application Programming Interface
ACID - Atomicity, Consistency, Isolation and Durability
BASE - Basically Available, Soft state, Eventual consistency
CRUD - Create Read Update Delete
DBMS - Database Management System
SME - Small and Medium sized Enterprises
VSE - Very small entity/enterprise

1 Introduction

To perform any software development there are choices that have to be made regarding what technology to use. These choices can have significant effects on a company's productivity, but it is difficult to do the comprehensive evaluation of available options that is needed in order to select the most suitable technology (Farshidi et al. 2018). A subset of technology decisions that has to be made are the choices related to what database model and database management system (DBMS) to use, where making the wrong choice could prove costly (Hassan, 2021). There is research describing general guidelines on factors to consider when making these decisions and proposed methods for the database selection process. However, Very Small Entities (VSEs) as defined by Laporte et al. (2008), have different characteristics than larger entities, which suggests that the best way to approach a database selection process might also be dissimilar.

This study explores the database model selection process of VSE's within the software development industry. This is done through a case study of a representative VSE, where their database model selection process is compared to a theoretical database model selection process. This includes comparing the factors taken into consideration when making the decision, and the actual steps taken in the selection process itself.

The case in this study is focused on the database model selection in a specific VSE, which is engaged in software development on a multitude of platforms. For example, the studied company develops Virtual Reality (VR) applications, web applications, mobile apps, and games. This variety of activities and projects can put specific requirements on the technology selection. This study, however, is limited to the database model selection process.

1.1 Background

In most software applications data needs to be stored persistently, which in turn requires a database. According to Hassan (2021), it is important to carefully select the appropriate database model for the intended purpose, and that many factors should be taken into consideration when making this selection. However, making any technology selection that is aimed at picking the most suitable choice, according to preferences and requirements, is a difficult task (Farshidi et al., 2018). Farshidi et al. (2018) also argues that the selection of efficient and cost-effective database technology is a crucial challenge for software producing organizations. Researchers have presented a variety of techniques and tools to solve different technology selection problems, but the majority of approaches use methods that are not scalable (Farshidi et al. 2018). These methods are costly and not universally applicable, and furthermore, technology selection decisions are often made on an ad hoc basis, without using predetermined selection models or methods (Farshidi et al. 2018).

Small companies may have different considerations and priorities than a larger organization regarding the challenges they face, but they may also have opportunities that larger organizations do not have. For example, one significant difference between VSE's and larger enterprises is the flexibility with the smaller structure of a VSE (Laporte et al., 2008). Small companies developing software have an important role in the economy and components developed by them are often included in larger companies' software (Laporte et al., 2008). In the EU's information and communication services sector roughly 95% of the companies were Micro enterprises, and together employed about 22% of the workforce in 2020 (Eurostat, 2023). There are many different definitions of small enterprises currently in use. Previously mentioned "Micro" enterprises are defined as those having less than 10 employees, and the term is included in the definition of Small and Medium sized enterprises (SMEs), which is part of EU legislation (European Commission, 2020, p. 11). A Very Small Entity (VSE) is defined by ISO (2016) as an enterprise, organization, department, or a project having up to 25 people. This is very similar to the definition used by Laporte et al. (2008), but Laporte et al. (2008) use the abbreviation to mean Very Small Enterprises. For the intents and purposes of this study, both meanings of the abbreviation VSE work equally well.

1.2 Problematization

There are many aspects to consider when choosing a database model. Choosing the wrong solution can cause problems in the long run, for example, if it does not scale appropriately. Therefore it is important that the database model selected is right for the current and future requirements. To increase the chances of selecting the most suitable database solution, using the right process when making that decision is a good idea. This can save time and resources (Roy-Hubara, 2019).

VSEs have characteristics that are different from those of larger organizations (Laporte et al., 2008). They, for example, often have fewer resources and are less process oriented and VSEs often have the view that processes are not made suited for them (Laporte et al., 2008). Therefore a VSE might be inclined to not use a structured approach for choosing a database model.

1.3 Purpose

In order to tackle the technology selection challenge, models and methods have been developed, but these do not necessarily conform to the limitations and characteristics of a VSE. The purpose of this study is to further understand how the database selection process is carried out in a VSE. In order to fulfill this purpose, this study aims to answer the following research questions (RQ):

RQ1: How is the database model selection process carried out in a VSE?
RQ2: Which factors are important to consider when selecting a database model in a VSE and how does this differ compared to theories regarding database model selection?
RQ3: How applicable are the existing theories regarding database model selection, on a VSE?

1.4 Delimitation

This study is delimited to focusing on the database model selection, which is a subset of technology selection. This study is further delimited to only studying how this is carried out in VSEs. Finally, this study is delimited to VSEs in the EU.

While the study is delimited to VSEs, the findings may still have broader implications. By examining a specific context in-depth, the study can contribute to the understanding of decision-making processes in VSEs and potentially offer insights that can be applied to other similar organizational settings.

1.5 Previous research

Hassan (2021) has researched how to make a database model selection based on the needs of the application at hand. Hassan (2021) compared the general differences between SQL and NoSQL databases and also reviewed the key differences between the NoSQL database models. The paper also covered important factors to take into consideration in the selection process. It concluded that it is important to thoroughly investigate these factors to make a good selection and that NoSQL, compared to relational databases, is increasingly becoming a viable option.

Roy-Hubara (2019) has created a structured method to be used for the database selection process. It was made through a case study with a design science approach where practitioners answered questionnaires and these were analyzed to gain insight into their situation so that all steps needed were included and formulated properly.

The study resulted in a proposed model divided into six steps, including gathering of requirements, dividing into fragments, and selecting and designing database models.

1.6 Research contribution

This study aims to contribute to the existing theory on the database selection process, specifically focusing on VSEs within the software development business. This study also aims to nuance the existing research by examining how well the pre-existing theories apply to VSEs.

1.7 Target audience

This study is targeted towards three target audiences. First, researchers who are looking to understand the decision making process for VSEs regarding selection of a database management system. This could be both those studying VSE's specifically, but also those wanting to understand any potential differences in the process between VSE's and SME's or larger enterprises. Secondly, VSEs who are approaching a database selection can benefit from this study, by gaining an understanding of the challenges, and possibilities they face, and also regarding the process for selecting a database management system. Finally, the studied company can benefit from deeper insight into their own database selection process.

2 Method

This study was conducted as a case study, focusing on understanding the database model selection process for software developing VSEs. This chapter describes and motivates the choice of research design, and describes how the data collection and data analysis was carried out. Lastly the validity and reliability of the study is discussed.

2.1 Research design

Yin (2009) suggests using a case study methodology when three criteria are met; the research questions regards the how or why of some phenomena, the researchers do not need control of behavioral events, and the study focuses on contemporary events. Research questions related to the how or why lend themselves to study via either a historical study, experiments, or case study. Historical studies are preferred when there is neither access or control. In other words, when the researcher only has access to documents and artifacts. Experiments are done when the researcher can manipulate behavior directly and precisely in a systematic manner. Case studies are preferred when examining contemporary events, but where the relevant behaviors are unable to be controlled or manipulated. It is similar to a history method but it adds direct observation of the events, and interviews as sources of evidence. These are unavailable in a history study. (Yin, 2009)

A case study can be defined as an empirical inquiry that investigates a contemporary phenomenon in depth, within its context, especially when the boundaries between the phenomenon and the context are not immediately clear. Furthermore, a case study deals with a technically distinct situation where there are more relevant variables than data points, and where the result relies on multiple sources of evidence and data. It is also benefiting from prior developments within the relevant theoretical fields, to guide data collection and analysis. This definition makes case studies best suited when a researcher wants to understand a real life phenomenon in depth, but where this understanding also comes with important context to the phenomena. (Yin, 2009)

Furthermore, Yin (2009) discusses four types of case study research designs, differentiated on two factors, the number of cases, and the number of embedded units analyzed. The four different research designs are: single-case holistic, single-case embedded, multiple-case holistic, and multiple-case embedded designs. There are several different rationales presented for each kind of case study research design. One rationale for single-case studies is that it is a representative or typical case. It may be, for example, a case study of a case that represents a typical project among many different projects, or a manufacturing firm that is typical of firms in the industry, a typical urban neighborhood, or a representative school. (Yin, 2009)

This study is a case study because it is exploring “the how and why” of a contemporary event, without the ability to control or manipulate variables, but with access to people involved. It was conducted using a single-case holistic design with the rationale that the studied case is a representative or typical case of a VSE within the software developer industry.

2.2 Data collection

The primary data for the case study was collected from the studied company’s website, from the developers presenting the company, and via two semi-structured group interview sessions with the developers at the studied company. Group interviews were chosen because of the two developers having different roles in relation to the database, with one of them having much more technical database knowledge than the other. The intention was to be able to obtain a more comprehensive understanding of the case in question by having the opportunity for the participants to complement and elaborate on each other's answers.

Semi-structured interviews were chosen because they can allow for more flexibility during the interviews, which in turn can give the opportunity to further explore the respondents' answers. Semi-structured interviews can also allow for a more conversational and relaxed interview style. Furthermore, Semi-structured interviews can provide the opportunity for the participant to express their views and experiences in their own words, rather than being constrained by predetermined response categories. (Adhabi & Anozie, 2017)

Since all the interview session participants were native Swedish speakers, that was the language chosen for the interview. This was also expected to enable a more natural and fluent interview, and give a more accurate description and understanding of the case. All interviews were recorded and transcribed in order to make analysis of the interviews easier. This was done through machine transcription followed by a thorough revision process that involved listening to the interview recordings and rectifying any errors encountered. After being transcribed, the interviews were translated to English and reviewed by the interview subjects. The interview subjects have been anonymized as “1” and “2” in both interview transcripts.

2.3 Data analysis

The collected data on how the studied company conducted their database selection process underwent an analysis based on three thematic areas derived from existing theory. The first theme involved comparing the factors considered during the selection process. The second theme encompassed examining the methodology employed in their database model selection. Lastly, the third theme focused on the characteristics of VSEs within the software development field.

To assess the significance placed by the company on various factors during their selection process, a comparison was made with the general guidelines provided by Hassan (2021) regarding factors to consider in database selection. Similarities and differences were identified, and the differing aspects were analyzed to provide potential explanations. In addition, the company's perspectives and opinions on the database selection method, as outlined by Roy-Hubara (2019), were gathered and analyzed, taking into account any disparities compared to their own approach. Moreover, the characteristics of the studied company were compared to the common attributes of VSEs described by Laporte et al. (2008) to validate its representation as a VSE.

2.4 Reliability and validity

Yin (2009, p. 76) states that there are four tests commonly used to assess the quality of any empirical social research. These are Construct validity, Internal validity, External validity and Reliability. Yin (2009, p. 76-77) has provided a number of tactics to handle each of these tests as well as what phase each tactic is to be applied in.

In order to meet the criteria of construct validity Yin (2009) provides three tactics: first, using multiple sources of evidence, secondly, establishing a chain of evidence, and thirdly, handing the draft of the case study to be reviewed by key informants.

Since this study examines one company and their database model selection process, there is only their version of events to study, and thus only them as sources of evidence. Secondly, the chain of events in this case is also relatively simple, and there is only a narrow set of evidence required to get a good picture of the case. Furthermore, the transcripts from the interviews, and the case study report itself has been continuously available to the studied company for feedback and confirmation that the explanation of events and case facts are correct. Unfortunately no database selection documentation could be analyzed because the process was not documented, which is often the case for VSEs (Laporte et at., 2008).

The criteria of internal validity is mainly a concern when trying to explain how and why event x led to event y. If there is an incorrect conclusion that there is a causal relationship between x and y without awareness of a potential third factor z which may have caused y, the research design has failed to secure internal validity. However, this is not applicable to descriptive or exploratory studies, which are not concerned with that kind of causal consideration. The concern regarding internal validity in case studies becomes relevant because of inferences made in the instances where an event cannot be directly observed. Based on the evidence collected during the case study, an investigator might make the inference that a particular event occurred as a result of some earlier event. (Yin, 2009)

This study does not attempt to present any inference regarding the causes of events, but instead makes a comparison between the studied case and the theoretical understanding.

The external validity regards to which extent the study's findings are generalizable beyond the immediate case and whether or not it will replicate. Case studies rely on what Yin (2009) calls analytic generalization, as opposed to survey research which relies on statistical generalization. In analytical generalization the investigator strives to generalize a set of results to some broader theory. (Yin, 2009)

The External validity of this study was handled by ensuring that the studied company was representative of VSEs. This was done through asking questions about their characteristics and comparing this data to the common characteristics of VSEs that have been found in previous research. The External validity would arguably have been better if a sample of VSEs within the software development industry had been selected, rather than just one organization, but this could not have been done within the time frame of this study.

Reliability refers to the ability for later investigators to conduct the same study again with the same findings and conclusions. The goal is to minimize errors and biases in a study. One prerequisite is documenting the processes and procedures followed. The general way to approach the reliability problem is to make as many steps as operational as possible and conduct the research so that an auditor, in principle, could repeat the procedures and arrive at the same results. (Yin, 2009)

The main steps taken in order to ensure high reliability in this study is to have a clearly explained research method, providing the interview guides and the transcribed interviews.

2.5 Method reflection

For the purpose of transparency, it is important to acknowledge that one of the researchers involved in this study had prior connections with the individuals interviewed from the company. This relationship could potentially raise concerns about bias or compromised objectivity in the research process. Efforts were made to mitigate the potential influence of these previous connections on the interview results. The researcher approached the interviews with a neutral mindset, striving to maintain objectivity throughout the entire process. Furthermore, it is worth noting that there was little or nothing to gain for the company by deliberately portraying themselves in a certain way. It was in the best interest of the company to provide an accurate description of their process since the result of the study would not be of much value to them otherwise.

This study was conducted as a retrospective study, involving data collection on two database selection processes: the first one was carried out in 2019 and the second in 2021. Conducting retrospective studies poses certain challenges that should be acknowledged. One of the primary concerns is the potential for recall bias. Since the database selection process took place some time ago, participants may struggle to accurately remember specific details, leading to inaccuracies or incomplete information. This limitation could potentially impact the reliability and validity of the findings. Furthermore, the use of group interviews for data collection introduces the possibility of groupthink. Group dynamics and conformity pressures within the interview setting might influence participants' recollections and lead to shared beliefs or perceptions that may not align with individual perspectives. These factors should be considered when interpreting the results and making conclusions based on this retrospective study.

While the number of interview subjects in this study may initially appear too small to base any conclusions on, it is important to consider the unique context in which the research was conducted. The study involves only two individuals who hold exclusive responsibility for database selection within their company. As the only individuals involved in this process, their insights and perspectives represent the entirety of the population under investigation. While the sample size may be small, the comprehensive exploration of these two individuals' experiences and decision-making processes provides valuable insights that can contribute to a deeper understanding of the database selection process within this specific organizational context.

3 Theory

This section presents and describes the theories which underline this study. First, the database model selection process is explained, focusing on both which factors that should be evaluated, and what steps should be taken during a selection process. Secondly, the comparison between when SQL and when NoSQL is best suited is presented. Finally, the specific characteristics of a VSE are presented.

3.1 Database model selection process

According to Hassan (2021) there are several factors to consider when deciding what database model is most suitable. These include; what type of data will be stored, what amount of data will be stored, schema characteristics, software and hosting related costs, transaction amount, and transaction frequency. Hassan (2021) also states that a general guideline is to choose SQL for a small or medium size application and to choose NoSQL for a large application that needs to handle large amounts of data. However, it is very important to identify the specific requirements for the application at hand and that is an essential part of the process of selecting an appropriate database model. (Hassan, 2021)

Roy-Hubara (2019) presents the following six step method intended for selecting and designing a database model:

1. Gather and specify the data related requirements and express them using a conceptual data model. In this work we use the UML class diagram, chosen based on its widespread use.
2. Gather and specify the functional requirements that are related to database operations, i.e., data retrievals and updates operations. Hereafter we call them queries.
3. Gather and specify the non-functional requirements (NFRs) that are related to the data requirements and the queries.
4. Based on the above, the method considers dividing the conceptual data model into fragments, each of which has different characterizations (such as different access frequency, different performance requirements, and different consistency requirements).
5. Select the most suitable database model/system for each fragment. This will be based on a general-purpose pre-defined profile for each database model. A predefined profile consists of a set of non-functional properties associated with each database model.
6. Design the selected database model/system for the different fragments.

(Roy-Hubara, 2019, p. 73-74)

This method takes factors into account such as data related requirements, functional requirements and non-functional requirements. The result of the method should be a recommendation of what appropriate model/models to use for the database and a recommended design. (Roy-Hubara, 2019)

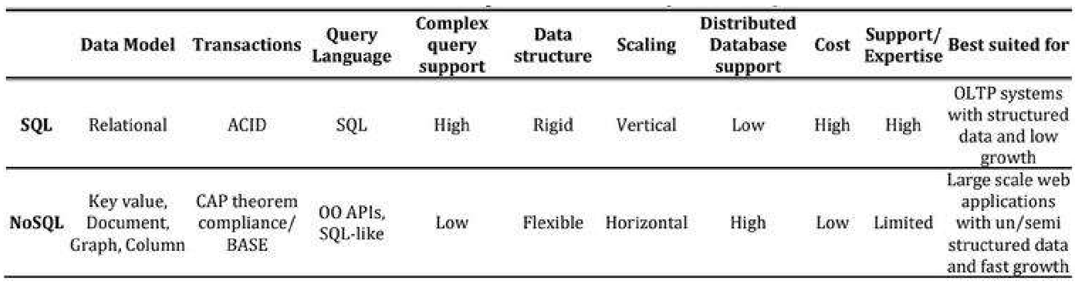
3.2 Comparison between SQL and NoSQL databases

Nisa (2018) states that the relational model is useful in regards to reliability, but when it comes to more modern applications, larger amounts of data, and when the data is unstructured non-relational models are more usable. In the relational model, data is stored in tables made up of rows and columns. Because this data needs to be reliable, it needs to adhere to the ACID properties. These are Atomicity, Consistency, Isolation, and Durability. Atomicity means that all transactions have to be fully performed for the database update to go through. Consistency is to have transactions carried out accurately in alignment with the database constraints and that what is done in past transactions can be seen in future transactions. Isolation means that even if queries are made simultaneously they do not affect the outcome of each other. Durability ensures that changes to the database are stored permanently. (Nisa, 2018)

In the non relational model, data is stored in various formats, such as XML or key value pairs. NoSQL data models do not ensure the ACID properties, but instead it offers BASE properties. This stands for Basically Available, Soft state, and Eventual consistency. Basically Available means that the database is almost always accessible. Soft state means that the database might not be consistent at all times. Eventual consistency points at the property of not being consistent all the time but that consistency is reached over time. (Hassan, 2021)

The problems with using relational databases for many modern applications are, for example, lack of high scalability, expensive set-up and maintenance, highly complex SQL when data is unstructured, lack of easy encapsulation of data due to the storage in tables, and difficulty sharing information between databases. Many of the problems are handled with NoSQL. Advantages of NoSQL over relational databases are for example that they are easily scalable, lower need for database administrators, ability to store structured or unstructured data, ability to program databases to handle hardware failures, higher speed, higher efficiency, and greater flexibility. Some of the disadvantages with NoSQL are a lack of a standardized query language, more difficult maintenance, and some databases do not follow ACID properties. (Nisa, 2018)

Mihai (2020) provides an overview of the differences between SQL and NoSQL databases, summarized in *figure 1*. One difference not covered above is SQL databases' high support for complex queries which NoSQL databases do not have due to them not including things like relationships and foreign keys. Mihai (2020) also explains the reason why SQL and NoSQL databases have different scaling capabilities. Unlike relational databases that can be scaled mainly vertically, by increasing the capacities of servers, NoSQL databases can easily be distributed on multiple servers, thus offering the possibility to add new servers in the system architecture as needed.



	Data Model	Transactions	Query Language	Complex query support	Data structure	Scaling	Distributed Database support	Cost	Support/ Expertise	Best suited for
SQL	Relational	ACID	SQL	High	Rigid	Vertical	Low	High	High	OLTP systems with structured data and low growth
NoSQL	Key value, Document, Graph, Column	CAP theorem compliance/ BASE	OO APIs, SQL-like	Low	Flexible	Horizontal	High	Low	Limited	Large scale web applications with un/semi structured data and fast growth

Figure 1. General SQL and NoSQL comparison (Mihai, 2020).

NoSQL databases can be classified into four different types, depending on the model used for data representation and storage. These are key-value stores, document stores, graph stores, and column stores. (Mihai, 2020)

Key-value stores have a key, or identifier, associated with a value. The value can contain an array or object, or a simple string or integer. The data can only be accessed through the key so it is not possible to search for a certain value. They can perform well in use cases like retrieving information for an online shopping cart or user profile configuration. (Mihai, 2020)

Document stores have their data in document types like XML and JSON. There is a lot of flexibility in the structure of data in these documents. They can contain different data types, arrays and even nested objects. Data can be retrieved by querying the names of the attributes, the values or the id of the whole document. The suitability of document stores differs slightly depending on what document format is used but JSON stores are well suited for web applications. (Mihai, 2020)

Graph stores have a graph structure which means that data entities are represented as nodes which can have relationships that are represented as edges. Both the entities and their relationships can have properties. Graph stores are efficient when handling data with complex relationships and are suitable when analysis of the relationships are as important as that of the entities. (Mihai, 2020)

In a Column store the data from one column (attribute) is stored together on disk instead of having the rows stored together like in a relational database. This makes retrieving all data on one column faster and retrieving all data from one row slower. Therefore Column stores are suitable for data that is used for analytic tasks where you want aggregation of a few attributes over a large number of rows. (Mihai, 2020)

A summary of the differences and similarities between the relational model and the four NoSQL types separately can be found in *table 1* below. It is often the case that a company has to use more than one database model to cover all their needs. Because of this it has in recent years become more common with DBMSs having support for more than one database model so that they can be used as multi-model systems. (Mihai, 2020)

Table 1: Summary of the main characteristics of relational databases and NoSQL stores. (Mihai, 2020)

	Relational	**Key-Value**	**Document**	**Graph**	**Columns**
Base data structure	Tables	Key-value pairs	XML or JSON Files	Nodes	Tables and Columns
Data Schema	Rigid	No Schema	Flexible	Flexible	Flexible
Support for joins	Yes	No	Yes	Yes	No
Ad-hoc query	Yes	No	Yes	Yes	Yes
Best suited when	Complex queries and frequent CRUD operations on structured and slow growth data are needed	High speed and highly scalable caches for applications are needed	Data can easily be interpreted as documents with rapid and constant growth	Relationships between entities are more important than the entities themselves	Analytical needs on very high volume of data are necessary

According to Mihai (2020), Relational databases are best suited when complex queries and frequent operations on structured and slow growing data are needed. Relational databases are the only ones in the comparison that require a rigid schema. Key-value stores are best suited when speed and scalable caches are needed. Document type databases are most suited when data can be interpreted as documents with rapid and constant growth. Graph databases are best suited when the relationships between the data is as important as the data itself. And column data stores are best suited when there are high analytical needs on very high volumes of data necessary. (Mihai, 2020)

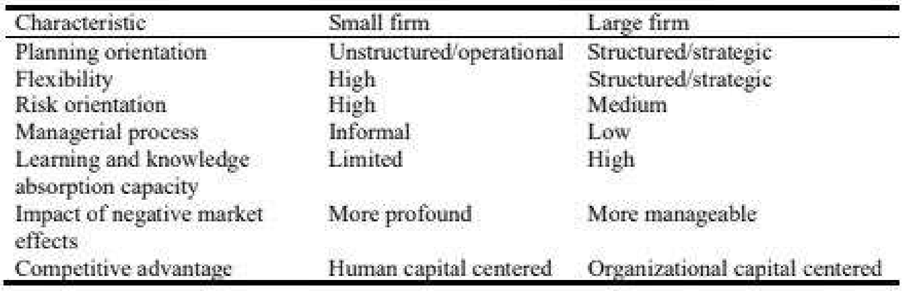
Mihai (2020) concludes that the relational model and the NoSQL models are not mutually exclusive, but that they are complementary and each has advantages and disadvantages in particular contexts. Each provides solutions for different scenarios, and both can coexist within a company. The relational model is more appropriate in the context of centralized, monolithic applications that use structured data, with reduced scaling needs that can be solved by vertical scaling, low data volume growth rate, but where transactions and complex joins are required. NoSQL models are more suitable for decentralized applications that use unstructured or semistructured data, where simple transactions and joins are all that is necessary, and where a high level of horizontal scaling will be needed, or where data volume increases rapidly, or where high and permanent availability is required. Thus, NoSQL tends to be a better option for modern web applications that need to store and process more complex and ever changing datasets in real time, where the data model needs to be flexible. The agility of NoSQL stores is another advantage for these databases, that in turn allows companies to enter the market faster and to make faster updates to their applications. Even though relational databases still have some well-defined scenarios where their usage is warranted, NoSQL stores offer many features that a relational database cannot offer without sacrificing a number of parameters such as speed or agility, and without greatly increasing cost. (Mihai, 2020)

3.2.1 Performance differences

There is a large body of research comparing the performance of SQL and NoSQL database solutions. In summary it can be stated that SQL and NoSQL have comparable performance but that NoSQL is faster in most cases. Wester & Fredriksson (2012) compared MySQL to MongoDB on speed using different sizes of data. They found that MongoDB was better on almost all the tests they ran, and concluded that there is a lot of time that could be won when handling larger data sizes. Parker et al. (2013) compared SQL using Microsoft SQL Server with MongoDB. They found that MongoDB had better runtime performance than SQL for inserts and updates, but SQL compared better when updating and querying non key attributes and handling aggregate queries. Rautmare & Bhalerao (2016) has compared the performance of MySQL and MongoDB, in the context of storing data from IoT devices. They found that in some scenarios, MongoDB was faster, but that MySQL responses were always stable, unlike MongoDB which had more varying response times depending on where in the database the record is stored. Ansari (2018) built two mock-up databases containing bank customer data using MySQL and MongoDB. Ansari compared the databases both on speed on CRUD-operations, as well as space efficiency and found that MongoDB outperformed MySQL in terms of speed overall. The biggest difference was with large amounts of entries. The MySQL database was more efficient with on-disk storage use than MongoDB.

3.3 Characteristics of Software Developing VSEs

The style of business is necessarily different between small entrepreneurial businesses and larger enterprises. *Figure 2* presents some of the characteristic differences between small firms and large firms.



Characteristic	Small firm	Large firm
Planning orientation	Unstructured/operational	Structured/strategic
Flexibility	High	Structured/strategic
Risk orientation	High	Medium
Managerial process	Informal	Low
Learning and knowledge absorption capacity	Limited	High
Impact of negative market effects	More profound	More manageable
Competitive advantage	Human capital centered	Organizational capital centered

Figure 2. Characteristic differences between large firms and small firms (Laporte et al. 2008).

The characteristics of VSEs are categorized by Laporte et al. (2008) into four categories: Finance, Customer, Internal Business Processes, and Learning and Growth.

VSEs are more financially vulnerable, since they are driven by cash flow and are dependent on project profits, which means they need to perform projects within budget. Furthermore, these budgets tend to be low, which in turn leads to fewer resources for activities such as; corrective post delivery maintenance, training, quality assurance, software reuse process, response to risks, process improvement and certification. (Laporte et al. 2008)

VSEs products typically have a single customer, where they are in charge of the management of the system, software integration, installation, and operation. It is often normal that the customer does not define quantitative quality requirements, and customer satisfaction depends on the fulfillment of specific requirements that may change during the project's course. Software development in very small companies is strongly human oriented, and communication between the parties is important, which is shown through a close relationship between all involved project members, including the customer. (Laporte et al. 2008)

The internal business processes of VSEs are usually aimed at developing custom software where the product is elaborated progressively, and which typically does not have strong relationships with other projects. Typically, most management processes such as human resource and infrastructure management, are performed through informal mechanisms, with the majority of communication, decision making, and problem resolution being performed face to face. (Laporte et al. 2008).

Finally, the learning and growth characteristics of VSEs are characterized by a lack of software process assessment and improvement, and a lack of human resources to engage in standardization. A negative perception regarding standards is common, as being made by and for large enterprises. (Laporte et al. 2008)

4 Results

In this chapter the results from the two interviews are presented (Full interview transcripts can be found in Appendix C and Appendix D). In the first part, RootPi and their database model selection process is described as well as the characteristics that make them a representable VSE. The second part elaborates on what factors were considered in the process. The third part contains a comparison between RootPi's method and a theoretical structured method for database model selection, and the company's views on the usability of the structured method in their context are also presented. Lastly their database model needs are summarized in order to do an evaluation of the outcome of their selection process.

4.1 Case description

The studied case focuses on RootPi and their selection of a database model, and how it suits their current and future needs.

RootPi is an independent software development VSE based in Lund. They work with a wide range of technologies and platforms, on a wide variety of applications. For example, they develop a VR simulation project in Unity, with 3D models made in Blender. The VR project is an external project for a medical university, aimed at enabling simulated training for the students. They also create AI-projects delivered on the web. These AI-projects often use a Python backend, which implements different AI models, and a front-end either written in HTML, CSS, and JavaScript, or some front-end framework, such as React or Vue. Finally, they also make mobile applications for a variety of purposes. They make most of their mobile apps for iOS and Android in the Dart framework Flutter. (RootPi, 2023)

Another nuance of RootPi's work and technological needs is that they do work both for their own publishing, and work on a contract or consulting basis. Nearly all of their technology selections are up to them.

Since RootPi work on a wide variety of projects, and frequently try new potential projects and ideas, they want to get a prototype up and running quickly. Since they use a wide variety of technologies, they want a solution that can work with all of them. In the first interview, the interviewee 1 states that: "[...] we started with Firebase in Flutter and then we realized quite quickly that their Firebase solution actually works for all coding languages. And then we actually changed our entire structure to use Firebase for everything. Because it's like only one thing to learn."

Since most of the RootPi projects have users from the EU, and since RootPi needs to store this user data for many of their projects, they need to be in compliance with General Data Protection Regulation (GDPR). GDPR is a privacy and security law that was put into effect by the EU in 2018. It affects all organizations that want to collect data related to citizens of the EU. The accountability of GDPR compliance lies on each organization, and violations can result in significant fines. (Wolford, 2022)

RootPi currently uses Firebase Cloud Firestore as their database solution. Firebase is a Google provided service that can handle backend functionality of an application without the developer having to write backend code (Firebase, n.d.). Cloud Firestore is a part of Firebase and is a NoSQL document database that is hosted on Google Cloud. It is not possible to host a Firebase database outside of Google Cloud (Firebase, n.d.).

RootPi's first DBMS was a MySQL solution running on a backend written in Laravel. This decision was made because one of the developers at the company had quite some experience from working at two different companies using Laravel. This previous experience and knowledge of both Laravel and MySQL made this the easiest thing to get up and running. Interviewee 1 explains it as: " I had experience from previously working at two separate companies using Laravel. So then it was as simple as that being what I knew and felt comfortable with. So it was the easiest thing to get going. I just used the tools I had and knew. That's kind of how I thought." And interviewee 2 adds: "Yes, absolutely. As I said, 1 was responsible for the database. So that's absolutely why we chose it." This solution ran on a server that also hosted their website, and it was relatively cheap. This solution was created because they had some projects where they needed to store data, including their company website, and some upcoming projects where they were later going to need to also store data online. So they just needed to get something going as a temporary solution, which could have been a permanent solution if it would have turned out that way. However, as they went along, they gradually realized that this was quite a problematic solution. The main problem was the time required to maintain a server running their own backend. This was one of the main reasons for the second DBMS selection: Firebase Cloud Firestore. Interviewee 1 says that "The reason why we changed was because it is quite a pain to write a backend on your own. On Firebase, you actually get the entire backend fully implemented. [...] So it's a pretty smooth and flexible way to collect data and it saves a lot of time for us by not having to implement a backend like we had to do in Laravel before."

The migration happened as they started developing mobile apps in Flutter, which is a framework for developing apps on multiple platforms in the Dart programming language. Flutter is developed by Google. When looking for data storage alternatives for Flutter apps they realized quite quickly that Firebase, which is also developed by Google, is able to work with Flutter smoothly. The main cause for trying out Firebase for their second Flutter app was that it was the most readily available. This made it very easy to test the solution and see how it would work and integrate with the project. After working with Firebase in that Flutter-app, they realized that the firebase solution works for all their projects. This led them to gradually change the entire structure and use Firebase everywhere. Interviewee 1 explains the migration as follows: "We first built a Flutter app with our own database. Then we built another Flutter app with Firebase. And then we revamped our entire website to use Firebase. So now nothing uses a proprietary database solution, but everything is via Firebase." and Interviewee 2 fills in with: "So there wasn't much research behind it. It was more like we found Firebase, looked around a bit, tested it and found this to be much smoother and easier. And for me, who is less experienced in MySQL and database management in that respect, it was even easier to just implement."

The needs regarding data storage have been largely the same over both database selections. The main requirement is to store and handle user data, and other data related to their company website.

4.1.1 VSE Characteristics

The following section provides the basis for stating that RootPi, the studied company, is a representative VSE. This is done by looking at the interview answers related to the characteristics of a VSE (Laporte et al., 2008) and evaluating whether the studied company has each of these characteristics. A summary can be found below in *table 2*.

Table 2. Summary of in what ways RootPi has the characteristics of a VSE.

VSE Characteristic	Does RootPi have this characteristic?
Unstructured/operational planning orientation	Yes, their weekly meetings are informal but the monthly meetings are a bit more formal.
High flexibility	Yes, they have no bureaucracy.
High risk orientation	Yes, they have big income fluctuations.
Informal managerial process	Yes, they use mostly undocumented face-to-face communication or other informal ways of communication.

Limited learning and knowledge absorption capacity	Yes, the company does not have resources for formal training.
Profound impact of negative market effects	Yes, they are financially dependent on single projects.
Human capital centered competitive advantage	Yes, their competitive advantages are centered around their competence rather than their organizational structure.

The studied company carries out their planning in an informal and somewhat unstructured way. They say “We try to have one meeting per week where we go over what we are doing, but these can quite often get mixed up in talk during lunch. So about once a month we try to have a slightly longer meeting to check where we are going.”. They use a meeting protocol but this tends to be unfrequently updated. They have annual goals and try to have a plan for the upcoming six months to one year.

The studied company answered that they think they are flexible, because of their small size. They have no bureaucracy, and there is no one to stop them from making decisions. Regarding project ideas one of the respondents said “We just ask each other if it sounds like a good idea and off we go.” If there is something new coming out, they can practically switch to it within a week. Regarding direct downsides they see no downsides of being flexible, but there are downsides related to being a small company.

One of the more prominent downsides is a lack of organizational structure, which leads them to having to handle all overhead processes themselves which takes time away from development. Related to this they believe that larger companies can have a broader range of competences than RootPi. But on the flip side of that, RootPi can be more specialized and have an advantage on specific things. Another upside of their small size is that they have exceptionally short time from conceptualization to live projects whenever they want to start a new project.

Because they are small, if a project goes badly it directly affects the company's finances in a big way. In other words, their income fluctuates a lot, because of their relative reliance on the results of each project. Furthermore, their small size also causes each project to have small budgets. The small budgets assigned to each project also lead to each follow-up effect that Laporte et al. (2008) mention: limited resources for Quality Assurance, corrective post delivery maintenance, and limited time for training and skill development. Because of their small scale, the studied company does not have room to employ anyone full time for tasks such as Quality

Assurance or maintenance. All of those tasks have to be done by the developers, which takes time away from developing potential new projects. Furthermore, there is no room for any courses or similar kinds of paid training, but they do not consider this a problem. Regarding Corrective Post Delivery Maintenance, interviewee 1 says: “It's not like we're leaving anything to rot. But it could definitely be better.” and interviewee 2 adds: “ Since we have a low budget, we can't afford to hire someone to do it while we do other things.”

4.2 Factors considered

The factors suggested by Hassan (2021) are presented in table 3 together with the perceived importance that the studied company places on the factor. In table 4 the factors considered by the company, but not mentioned by Hassan (2021) are listed.

Type of data, data structure, schema characteristics, transaction amount, and transaction frequency were all considered to some degree, but not thoroughly researched. Each of these factors were only researched to the level that the developers could be sure that they would be able to do the things they needed in a manner that is not overly complex. One reason for the relatively little focus on these factors is cited to be that they have relatively simple requirements regarding data and transactions. Of the factors that Hassan (2021) suggests that the studied company considered the cost was mentioned as the most important.

Table 3. Hassan (2021) factors considered by the studied company

Factor	**Importance placed on the factor**
Type of Data	Medium
Amount of Data	Low
Data Structure	Low
Schema Characteristics	Medium
Software and Hosting Related Cost	Medium
Transaction Amount	Low
Transaction Frequency	Low

Table 4. Factors considered by the company, not mentioned by Hassan (2021)

Factor	**Importance placed on the factor**
Total Costs / Time investment	Very high
Developer Experience	High

The type of data was one reason for switching to firestore. The fact that they needed to store personal data, which requires higher data security. This puts more pressure on the developers if they would have written their own backend solution, so the fact that Firestore offered a solution with, what they perceived as, high security was a strong reason for switching. Interviewee 1 puts it as “And we knew that we would save personal data because we needed a login system. So we took that into account when we looked around for solutions. That's kind of why we didn't want to write our own solution from scratch, but wanted to use some kind of framework or ready-made service. So that security is high and that you don't get, for example, SQL injections.”. They did not consider the factor type of data in the sense of what data types they would need to use and how that would affect performance of database transactions and such.

Regarding the amount of data to be stored, they said that they really did not need to think too much about it, since they are so far below the data amounts where that would be any kind of issue. In the words of interviewee 2: “At no stage so far have we had anything that has been on the verge of being overloaded. We haven't had enough data to really have to care about this yet.” However, the scalability of the solution was considered. The company quickly concluded that Firestore would cover all their current needs plus more. It was also expected that it would be much easier to scale in the future compared to the Laravel/MySQL solution, due to the fact that Firestore is cloud based and scaling is handled automatically. Scaling was not an immediate need or problem, but it was a factor that ultimately weighed in positively in the decision to switch from the Laravel solution to Firestore.

The factors transaction amount and transaction frequency were also factors that did not take much time for the company to research and consider, because they are quite far below levels of traffic where that can become problematic, or where they would think that it would become a significant factor. One point they make is that since Firebase starts charging for data at 50-thousand reads per day, and they are not up to those levels, it is clear that Firebase can handle much more than that.

In summary, the factors affecting performance; amount of data, transaction amount, and transaction frequency, were not very thoroughly researched before making the switch. But cursory research led them to believe that Firestore would provide adequate performance, since they found quite large projects running with Firestore as the main database. They mainly wanted to confirm that the performance would not be a negatively impacting factor after making the switch.

The factor schema characteristics were said to be important, mainly because they want an intuitive way to handle data that is linked to a user. Interviewee 1 describes it as: “We wanted there to be an intuitive way to retrieve the data that is linked to the user's data.” They also state that if Firestore would have been worse than SQL regarding this, they would have been less eager to make the switch.

The direct software and hosting cost was a factor considered important when making the evaluation. The direct software and hosting cost would be somewhat similar between the firebase solution and their original solution. This includes all of their use-cases, both scaling up an existing solution, and adding new products.

When asked about factors that they considered, but that was not included in the list of factors by Hassan (2021), they mention three factors: the total cost, the developer experience, and performance.

RootPi viewed the total cost as the most important factor, and that it also should include the cost in terms of development time. Interviewee 2 says that: “When it comes to cost. It is not only the cost in money that matters, but it is the cost in time that is the biggest, absolutely most important factor for us.” The developer experience and the ease of use also becomes somewhat related to the total cost, since this can increase or decrease the total development time. They exemplified this with a hypothetical solution that is very good in all of the other aspects, and it could be free, but if it has very bad documentation, then that weighs higher and it would not be a viable option.

The ease of development, and thus a lower total development cost, is cited as an important reason for switching. They consider Firestore much easier to use and less time consuming to implement than running their own backend on a server with a DBMS. When using Firestore, the backend is already handled and an Application Programming Interface (API) is provided for handling the data. This decreases the extra work which comes with handling their own backend and server solution. Another consideration that decreases the initial time investment was that Firestore is document based, which means that there is no requirement to model out all the data beforehand, which is a requirement in a relational database. In the second interview (Appendix D) Interviewee 1 put it as: “We remove as much time as possible in implementation because it works almost immediately with a good API. So that's probably the biggest factor.”. And in the first interview (Appendix C) this is how interviewee 1 describes Firestore: “On Firebase, you actually get the entire backend fully implemented. So you can just use their API to write data into their database and their database is also document based so you can actually submit anything to it. So you can sort of decide what your data is going to look like and then just submit it and it will be like that. So it's a pretty smooth and flexible way to collect data and it saves a lot of time for us by not having to implement a backend like we had to do in Laravel before.”

Furthermore, another aspect of the developer experience is that there is a large and active community, and good documentation. Which is explained by interviewee 1 as: “In case Firebase didn't have a good community, meaning that nothing is written about their solutions or that their documentation is bad. Then it would have been much more difficult to switch to a solution like Firebase. But luckily, Firebase has as good documentation as possible plus they have a lot of users. So if you encounter a problem, you can find solutions to those problems.” The fact that Firestore has good and reliable documentation, and a large user base that shares their problems and solutions, was viewed by the studied company as a strong positive that affected the choice to go to Firestore. This was also paired with a trust in Google as the developer of Firestore. Interviewee 2 adds: “We trusted Google as a company when we made the choice too, I guess. Partly that it was Flutter-bound, but also that we have confidence in Google as a company that if they provide a solution like this, it will be good enough for our requirements. It's the brand Google we trust, in part.”

4.3 Method used when selecting database model

The studied company did not use, nor were they familiar with, any structured model or method when doing any of their database selections.

Neither the selection of Firebase, nor the previous selection of Laravel and MySQL, were made in a highly structured manner. Their first selection was based mainly on what they already had experience with and were confident that they could get up and running. Interviewee 1 puts it as: ”I had experience from previously working at two separate companies using Laravel. So then it was as simple as that being what I knew and felt comfortable with. So it was the easiest thing to get going. I just used the tools I had and knew.” While making their second Flutter app they looked around for alternatives, and quite quickly found Firestore and that it seemed to be easily implemented. After testing Firestore, it seemed to fill all their current needs, and it has turned out to be quite a comprehensive solution for all their projects so far. Interviewee 1 explains it as “So when we looked around at how we could save data in a Flutter app, we realized quite quickly that Firebase, which is also developed by Google, is kind of integrated with Flutter in a very smooth way. So then we started with Firebase in Flutter and then we realized quite quickly that their Firebase solution actually works for all coding languages. And then we actually changed our entire structure to use Firebase for everything."

Table 5. Method for selecting database model at the studied company

Step according to method proposed by Roy-Hubara (2019)	**How this was carried out at the studied company**
Gather and specify the data related requirements and express them using a conceptual data model.	Data related requirements were considered, but no formal requirements specification was created.
Gather and specify the functional requirements that are related to database operations.	Functional requirements were considered, but no formal requirements specification was created.
Gather and specify the non-functional requirements that are related to the data requirements and the queries.	Non-functional requirements were considered, but no formal requirements specification was created.
Divide the conceptual data model into fragments, each of which has different characterizations.	No fragmentation was needed.
Select the most suitable database model/system for each fragment.	They selected a database model and system but did not compare multiple solutions.
Design the selected database model/system for the different fragments.	They designed their database model for their single fragment.

In general, they did not do much formal gathering of requirements, nor comparison with other solutions. They did not compile any kind of document with the specifications or data related requirements. However, they tried the Firestore solution, to make sure that it would work and that they would be able to store the data that they needed. This was elaborated on with the fact that they were small enough to have the luxury of testing things without doing much research beforehand. "We are small enough to have the luxury of testing stuff without having to do a lot of research."

Before the switch, they did not make any conceptual data model, but added that they usually sketch out how they want to store data before starting a new software project.

The step of gathering and specifying functional requirements related to the database operations was also carried out informally without creating artifacts or documentation. What they did overlaps a lot with what they did when gathering and specifying the data related requirements, as they tried Firestore to make sure everything that they wanted to do worked. Then they checked the Firebase API and saw that everything they think they would ever need can be done in Firebase. Interviewee 1 explains it as: “What we did was that we had a ready-made solution that worked on our servers. And before we switched to Firebase, we started testing with Firebase and storing the kind of data that we had stored before to see how it works. All types of data that we stored before work with Firebase.”

Non-functional requirements were also gathered and specified on an informal basis, without creating documentation or artifacts. What they did was that they checked out some aspects of Firebase, such as the security, and found that it did all the things they wanted it to do. For example, interviewee 1 says that: “We took a closer look at Firebase, for example on their security. They use server-side encryption and many standard things that you should have in your databases to make them secure.”

When asked, the studied company thinks that they could have benefited from following the laid out steps more formally. Especially if they would have had more possible options on the table. The reason given for not formally doing much comparison, or requirement collection, was that they found Firestore, checked if it could do what they needed it to do, made sure it would have the ability to scale, and then just switched. This is further motivated by them being a small company. Making an in depth formal comparison, collecting requirements, and thoroughly comparing it to the features of the technology can be very time consuming. Furthermore, they argue that finding the very best solution might not be worth the additional time investment, given that they can find something that is good enough with less time spent creating requirements, and making comparisons. Thus, they think making an informal comparison where they can make sure the tool will do what they need, and making sure that they avoid the major pitfalls is good enough to them, even if they don't find the most optimal solution. Interviewee 1 explains: “I think for us as such a small company. After all, it will be a cost in hours to make such a comparison. And how important is it for us to find the absolute best? It's not that important. As long as we find something that is good enough, it is much better for us to focus on developing our products. To make the comparison a bit informal in our scenario, I see it as better because it goes faster. You avoid the absolutely biggest traps. Even if it won't be optimal, it will be good enough.”

Regarding the questions about fragmentation of data models, the studied company answered that there was little reason for them to use multiple fragments, since it increases complexity and they have little to gain from it. “So far there has been no reason for us to have multiple fragments. It increases the complexity as well and we probably have nothing to gain from it.” The studied company did not compare several different alternatives, since they only compared Firestore to their previous Laravel/SQL solution. However, they did design the selected database model for the different fragments, even if they can be seen as only having one fragment. “We did not make a comparison between several different alternatives. We only compared the old solution against Firebase.”. And interviewee 2 adds: “It was perhaps a bit naive actually to only look at Firebase. But so far so good.”

In general, they thought the model presented by Roy-Hubara (2019) seemed like a sensible way to do things. However, they felt that it included a lot of things to do and that it appeared to be designed for larger companies. They thought that it would have been beneficial to have formal structures of what to do and what to consider when selecting a database model. Something like a step-by-step checklist where steps, or part of steps, that don’t apply could be skipped. Interviewee 1 says: “It is absolutely a sensible way to do it, to arrive at a system that makes sense.” and interviewee 2 adds: “I think there is a lot to do. I would guess that this is designed for large companies. The larger the company, the more applicable this method is.“. On the follow up question, interviewee 1 says: “It is great to have a formal structure of what is important to consider when choosing your database. So a step by step description of what to look at when making the choice. It makes sense.” and interviewee 2 adds: “ I agree. Because this really is a comprehensive guide. Which should work for all companies. And then if you come to the conclusion as a company that "we can ignore this part". Then you do it. But you can still have research at the bottom and know that you won't miss anything in a process like this.”

4.4 Summary of needs impacting the database model choice

The general needs from the general SQL and NoSQL comparison by Mihai (2020), for the studied company is best described as: Data Model agnostic, BASE transaction properties are sufficient, Query Language agnostic, no need for complex queries, preference for flexible data structure, preference for horizontal scaling, database distribution agnostic, very high preference for low cost, no need for paid support but very preferred to have good documentation and widespread community. This is summarized in *table 6*.

Table 6. Summary of the general needs for the studied company.

Factor or feature consideration
No need for complex queries.
Preference for a flexible data structure and schema.
Horizontal scaling requirement.
Preference for low cost.
No professional support required but community support strongly preferred.
No complex CRUD operations and data is not structured and slow growing.
High speed cache that scales well is not the main purpose of the database
Data can be interpreted as documents with rapid and constant growth.
The relationships between the entities are not more important than the entities themselves.
There is not a need for analytics on very high volumes of data.

The comparison, shown in *figure 1,* by Mihai (2020) shows that SQL is best suited for Online Transaction Processing systems with structured data and low growth, whereas NoSQL is best suited for large scale web applications with un/semistructured data and fast growth. Furthermore, the document based databases are often best suited when data can be interpreted as documents with rapid and constant growth. (Mihai, 2020)

5. Analysis and Discussion

This section discusses differences in factors considered and methodology between what the studied company did, and what is proposed by theories regarding database model selection.

5.1 Factors considered

The largest difference between the studied company's focus and the factors by Hassan (2021) are that the studied company valued the total cost as the, by far, most important factor. Hassan (2021), only has the direct costs in his factors, which presumably excludes costs arising from larger time investments. The explanation given by the studied company is that a small percentage change in transaction costs will amount to a very small change in the total cost. However, for a larger company with more transactions, a small change in percentage cost will amount to a large change in the total cost. The importance of the total developer time is also explained by the same dynamics. In a VSE every development hour is a larger percentage of the total cost than in a larger enterprise. This can be related to the fact presented by Laporte et al. (2008), that VSEs are more financially vulnerable.

One explanation for the low emphasis placed by the studied company on the factors Transaction amount and Transaction frequency as well as Amount of data and Data structure is related to cost efficiency of database transactions and related costs. At a large enterprise, small percentage efficiency improvements translate to a large total impact. In larger companies, cost efficiency of database transactions is more impactful on the total cost. For small companies, with few customers and few database transactions, an improvement in cost efficiency of the database transactions does not impact the total cost significantly. The studied company did however consider the software and hosting related cost an important factor but put most emphasis on the fixed costs and wanted a free tier solution.

Related to how it affects database transactions, the factor Type of data was not given much thought by the studied company. However, it was considered at the level of what information the data was going to contain. Because they were going to store personal data they knew they needed to have things like server-side encryption and other standard security features. They wanted to avoid having to implement them themselves and that is why they were looking for a DBMS solution which had these features out-of-the-box. This can be expected to be a tendency shared by most VSEs due to their limited budget for projects. The limited budgets for projects in VSEs is described by Laporte et al. (2008) as leading to fewer resources for several activities, such as quality assurance, and process improvements. Given that backend maintenance is not a direct revenue driver, the preference for a less time-intensive backend solution can also be explained by the lower budgets for projects within VSEs.

The Schema characteristics factor was also considered quite important but mostly from the viewpoint of developer experience, in the sense that they looked into the process of setting up and managing a schema for a new project to see how easy and intuitive it was.

Developer experience was a factor that Hassan (2021) did not bring up, but that the studied company put significant emphasis on, related to the selected technology. They pointed to things like how easy something is to learn, how fast and comfortable it is to develop in, and the development time that can be saved using a technology that provides a better developer experience. Less time needed for database related activities enables the studied company to focus more time on their main income generating activities.

5.2 Method used when selecting database model

It was expected to find that the studied company did not follow a structured method for the database model selection. The reason for this is mainly because of limited time for the process. This can also be related to the characteristic explained by Laporte et al. (2008), that VSEs have smaller budgets for projects. Not because of pressure from deadlines, but because the selection process or the database implementation is not directly generating income for the company, and therefore a larger time investment is not easily motivated. Also, the gains from doing a thorough comparison of all or most of the available options, to find the best possible solution, was estimated by the company to be low compared to the cost of the time investment.

Even though the studied company made a first selection of a backend, DBMS and database model that they were unhappy with, they managed to switch to a solution they felt more pleased with. The flexibility, unstructured planning, and informal managerial processes played a significant role in helping the studied company correct this initial decision. Flexibility, unstructured planning and informal managerial processes are key characteristic differences between large firms and VSEs described by Laporte et al. (2008).

5.3 Evaluation of database model choice

Looking at the choice of Firebase Cloud Firestore, under the lens of Mihai (2020)'s comparison, we can evaluate the choice based on the needs and the use case presented by the studied company. When comparing the general requirements for the studied company, to the general comparison by Mihai (2020), NoSQL is determined to be a good choice.

First of all, because the studied company values the flexibility of not having to pre-define a data schema highly, this leans toward not using an SQL solution. This is because when using an SQL database, all data has to be modeled beforehand, and it is hard to change it after the fact (Mihai, 2020). With a NoSQL solution, there is no need for a schema, and data can be much more flexible (Mihai, 2020). Another point that heavily weighs in favor of a NoSQL solution is the fact that the studied company wants the database to scale horizontally, and they have little need for complex queries. Furthermore, one of the main reasons why NoSQL, and more specifically, a document based database model is the best suited, is that the data can be interpreted as documents (Mihai, 2020). Finally, the studied company expects rapid and constant growth, both in aggregate, and in each of the projects they start. This also motivates the document based database model (Mihai, 2020).

This is contrasted with other types of NoSQL databases. In the case where the main database purpose is to provide a high speed and highly scalable cache, then a key-value store is the most suitable. In the case where the relations between entities are more important than the entities themselves, then a graph based database model is most suitable. In the case where there is a need for analytics on a very high volume of data, then a column data store is the recommended database model. (Mihai, 2020) Neither of these facts are true in the studied case, which further motivates the opinion that a document based NoSQL database is the right choice for the studied company.

Regarding performance, the studies Wester & Fredriksson (2012), Parker et al. (2013), Rautmare & Bhalerao (2016), and Ansari (2018) show that both SQL and NoSQL are quite performant, but that there is better performance scalability when using NoSQL. The performance of SQL is also heavily dependent on indexation. The general outcome is that there is a slight performance advantage when using NoSQL.

Given these comparisons and the data from the studied company, they have made a good selection, based on their current and projected needs. Their first selection was not optimal, which they also realized. However, they managed to make the switch from that initial selection of SQL to their current NoSQL solution.

6. Conclusions

This study has sought to deepen the understanding of the database selection process within a VSE context, since the available methods may not account for the specific characteristics of a VSE. In this section, the results are presented in relation to the initial research questions.

RQ1: *How is the database model selection process carried out in a VSE?*

In summary, the database model selection process in a VSE is carried out informally to a large degree. The selection is mainly done without following a structured method, and through trial and error, rather than planning carefully. The studied company did not document their process or any requirement specification, and made their technology decisions largely based on what was accessible to them; aiming for getting a good enough solution fast rather than making sure they found the optimal one.

RQ2: *Which factors are important to consider when selecting a database model in a VSE and how does this differ compared to theories regarding database model selection?*

As a result of this study we can conclude that VSEs have different priorities, regarding which factors are most important, compared to larger enterprises and what is brought up by existing theory regarding the general case. The most important aspect for VSEs is the total cost, which includes total development time. The importance of total cost, including development time, could lead smaller software development enterprises to weigh the developer experience as more important, than when a larger enterprise selects technology. This is because a part of developer experience is the overall usability of a technology which can lead to more efficient development and shorten certain parts of the development process.

Factors like Transaction amount and Amount of data are not very important for smaller scale implementations and do not have to be investigated in depth in the database model selection process of a VSE. This is because with relatively low volumes of transactions and with small amounts of data to store, most database models paired with most of the popular DBMS options can handle these requirements.

RQ3: *How applicable are the existing theories regarding database model selection, on a VSE?*

The main conclusion regarding the use of methods for selecting technologies, and more specifically, database models, is that a VSE has higher flexibility which in turn lets them have greater odds of parrying a poor decision regarding technology. Using a model that is designed to help with technology selection is advisable, but it should not over-encumber the organization, so that more energy and time is put into following a model, than is put into producing valuable products and services.

The method examined in this study gives good outlines, but could be too cumbersome for a VSE. One conclusion is that following something more akin to a checklist could be beneficial for a VSE in their technology selection. For VSEs, a good enough choice is probably the most important goal. Spending a little bit of time to avoid making an outright bad decision is likely time very well spent, but then spending way more time in order to find a perfectly optimal choice is probably a poor use of time. Getting the wheels off the ground and getting to work on projects that lead to income, rather than optimizing decisions early, should be the top priority for VSEs.

These conclusions regarding methods used, are motivated by the fact that even though the studied company made a poor initial decision, they had the flexibility to make a new technology decision. After evaluating the new decision they could migrate everything over to this new decision, without having to close down projects, or making huge sacrifices.

6.1 Further research

Based on the findings in this study, some suggestions for further research are presented below.

Technology selection other than database models in VSEs is an area that could be explored further. This could help VSEs make better decisions regarding their technologies. Another suggestion is that a database model selection method tailored for VSEs could be created and tested through action research or design science research methodology. Alternatively a modification of the method presented by Roy-Hubara (2019) could be tested and evaluated. Another topic that could be further explored is how the developer experience factors in when making a technology selection. Since the studied company put such a large emphasis on the developer experience and its impact on the total cost of development, it is a potentially valuable area of further research.

References

Adhabi, E., & Anozie, C. B. (2017). Literature review for the type of interview in qualitative research. *International Journal of Education*, *9*(3), 86-97.

Ansari, H. (2018). Performance Comparison of Two Database Management Systems MySQL vs MongoDB [Bachelor's thesis]. DiVA. http://urn.kb.se/resolve?urn=urn:nbn:se:umu:diva-155398

European Commission (2020). *User guide to the SME definition,* Publications Office. https://doi.org/*10.2873/255862*

Farshidi, S., Jansen, S., De Jong, R., & Brinkkemper, S. (2018). A decision support system for software technology selection. *Journal of Decision Systems*, *27*, 98–110. https://doi.org/10.1080/12460125.2018.1464821

Firebase (n.d.). *Cloud Firestore*. https://firebase.google.com/docs/firestore

Hassan, M. A. (2021). Relational and NoSQL Databases: The Appropriate Database Model Choice. *2021 22nd International Arab Conference on Information Technology (ACIT),* 1–6. https://doi.org/10.1109/ACIT53391.2021.9677042

International Organization for Standardization [ISO]. (2016). *Systems and software engineering - Lifecycle profiles for Very Small Entities (VSEs).* ISO/IEC TR 29110-1:2016(E)

Laporte, C. Y., Alexandre, S., & O'Connor, R. (2008). A software engineering lifecycle standard for very small enterprises. *EuroSPI 2008 - European Systems and Software Process Improvement and Innovation, 3-5 September 2008, Dublin, Ireland. ISBN 978-3-540-85936-9*.

Mihai, G. (2020). Comparison between Relational and NoSQL Databases. *Annals of Dunarea de Jos University. Fascicle I : Economics and Applied Informatics*, *26*(3), 38–42. https://doi.org/10.35219/eai15840409134

Nisa, B. (2018). A Comparison between Relational Databases and NoSQL Databases. *International Journal of Trend in Scientific Research and Development, 2(3), 845-848*. https://doi.org/10.31142/ijtsrd11214

Parker, Z., Poe, S., & Vrbsky, S. V. (2013). Comparing NoSQL MongoDB to an SQL DB. *Proceedings of the 51st ACM Southeast Conference, 5,* 1-6. Association for Computing Machinery. https://doi.org/10.1145/2498328.2500047

Rautmare, S., & Bhalerao, D. M. (2016). MySQL and NoSQL database comparison for IoT application. *2016 IEEE International Conference on Advances in Computer Applications (ICACA)*, 235-238. https://doi.org/10.1109/icaca.2016.7887957

RootPi (2023). *This is us.* https://rootpi.xyz/

Roy-Hubara, N. (2019). The Quest for a Database Selection and Design Method. *International Conference on Advanced Information Systems Engineering.*

Wester, A., & Fredriksson, O. (2012). Jämförelse av Mysql och MongoDb [Bachelor's thesis]. DiVA. http://urn.kb.se/resolve?urn=urn:nbn:se:bth-2310

Wolford, B. (2022). *What is GDPR, the EU's new data protection law?* GDPR.eu. https://gdpr.eu/what-is-gdpr/

Yin, R.K. (2009). *Case study research: design and methods*. (4. ed.) London: SAGE.

Appendix A. Interview guide 1

Information

1. Information regarding the study
2. Information regarding the planned purpose, structure of the interview, and ask consent for recording.
3. Information regarding anonymity, freedom to skip questions, and data usage
4. Information regarding that they will get to see the transcripts afterwards and confirm if it looks correct

Main interview section

5. What is your DBMS and how did you select it?
    a. Did you follow any structured method? Why/why not?

6. What were the most important factors when deciding on a database solution for you?
    a. Technical, Scaling, other?
    b. Developer experience? Fast and smooth development?
    c. Estimated costs/pricing?
    d. Community support?

7. What were your needs regarding a database/data storage?
    a. What kind of data did you mostly store?
    b. What kind of operations were most common for you?
    c. What were your demands regarding performance?

8. Other questions.
    a. What are some pros and cons with your current DBMS?

Appendix B. Interview guide 2

Factors

1. Which of the following factors did you consider in the process of making your database decision?
    a. What type of data you would store
    b. What amount of data you would store
    c. Schema characteristics *(how/if the data is linked together)*
    d. Software and hosting related costs
    e. Transaction amount & transaction frequency
    f. Is there any factor that you have in mind that we haven't asked about?

Database Selection Model

2. Are you familiar with Roy-Hubaras' process for selecting a database, or with any other structured method for database selection?

3. Roy-Hubara (2019) suggests a process/method for database selection, we would like to go over it step by step and hear:
    a. To which extent you did each step, what you think about it.
    b. If you didn't do it:
        i. Why not?
        ii. Do you think it would have been beneficial if you would have done it?

    1. Gather and specify the data related requirements and express them using a conceptual data model.
    2. Gather and specify the functional requirements that are related to database operations.
    3. Gather and specify the non-functional requirements that are related to the data requirements and the queries.
    4. Based on the above, the method considers dividing the conceptual data model into fragments, each of which has different characterizations (such as different access frequency, different performance requirements, and different consistency requirements).
    5. Select the most suitable database model/system for each fragment. This will be based on a general-purpose pre-defined profile for each database model. A predefined profile consists of a set of non-functional properties associated with each database model.
    6. Design the selected database model/system for the different fragments.

    c. Considering this method as a whole. Do you think it is well suited for a company of your size?

Characteristics

4. Which of the following traits does your company have?
    a. Is your planning carried out mostly face-to-face, informally and unstructured?
    b. Would you say that your company has a human capital centered competitive advantage? (rather than organizational centered)
    c. Would you say that your company is more flexible due to its size?
        i. Can you elaborate on this?
        ii. Do you find a lack of organizational structure a downside?
        iii. Are there more strengths or positive aspects which come as a result of you being a small business?
    d. Are you financially vulnerable to one individual project having issues?
    e. Would you say that your projects tend to have low budgets?
        i. Does this lead to limited resources for:
            1. Quality assurance activities?
            2. Corrective post delivery maintenance?
            3. Training?

Appendix C. Interview 1 translated transcript

2023-03-27

What is your DBMS and how did you select it?
1: Previously we had our own database solution written in Laravel and now we have switched to Firebase, where we use Firestore. The reason why we changed was because it is quite a pain to write a backend on your own. On Firebase, you actually get the entire backend fully implemented. So you can just use their API to write data into their database and their database is also document based so you can actually submit anything to it. So you can sort of decide what your data is going to look like and then just submit it and it will be like that. So it's a pretty smooth and flexible way to collect data and it saves a lot of time for us by not having to implement a backend like we had to do in Laravel before.

Did you have other options you looked at?
1: What happened was that we started developing Flutter apps. And Flutter is developed by Google. So when we looked around at how we could save data in a Flutter app, we realized quite quickly that Firebase, which is also developed by Google, is kind of integrated with Flutter in a very smooth way. So then we started with Firebase in Flutter and then we realized quite quickly that their Firebase solution actually works for all coding languages. And then we actually changed our entire structure to use Firebase for everything. Because it's like only one thing to learn. Everything works exactly the same and we can sit and chat with it. They also have quite a few different solutions. So the database is there. But they also have, for example, login (SSO) so you can authenticate with Google accounts, Facebook accounts and everything via their Firebase API. So it means that whole products from our side can use new technology. Which is quite nice. Without having to use a lot of different things for different languages.

I interpret that to mean that you didn't make a list of requirements and needs or something like that?
2: No, that was like the best part. It was the most readily available back then. And then we tested using it and it just seemed to work for all the needs we had at the time. And then it turned out to be a pretty comprehensive solution for everything we've done so far.
1: Before that we had MySQL which we implemented ourselves. And the only important thing for us is that it should be free. And since we're so small, MySQL if we implement it ourselves, it's free. Although we still have to pay server costs to keep it available online 24/7 as well. But with Firebase, when you pay for how much you

use, we still don't reach traffic levels that we need to pay for. So then the Firebase solution actually turned out to be free plus it meant less work for us. When we realized that it was a pretty simple choice to switch.

And you made a gradual migration?
1: Exactly. We first built a Flutter app with our own database. Then we built another Flutter app with Firebase. And then we revamped our entire website to use Firebase. So now nothing uses a proprietary database solution, but everything is via Firebase.
2: So there wasn't much research behind it. It was more like we found Firebase, looked around a bit, tested it and found this to be much smoother and easier. And for me, who is less experienced in MySQL and database management in that respect, it was even easier to just implement. And as 1 mentioned before, you don't need to know the structure in advance, you can just create new elements as you go.

We have touched on this already, but we're wondering what factors you took into account when making your database choice.
Did you think about scalability?
1: Yes, so that wasn't a reason why we chose Firebase. Or maybe you could say it was. Because we know that Firebase is developed by Google. They will have security. Plus the scaling is like, as soon as you get more users, they scale up your system automatically. So there we have the scalability, which we didn't have with our own solution. So we knew it kind of covered what we had before plus a lot more.

So it was something that you looked at after you tried Firebase. Then you checked, for example, that it would scale?
1: Yes, exactly. So we saw right away when we started looking at Firebase that it was scalable.
2: Yes, exactly. It wasn't a problem that we had and was like, oh we have to switch up now. Or we have to change the database because we have to scale it up. But it was just, when we found Firebase, it was just a bonus point that Google was handling it. And we know it will work for companies that are small like us, but also for larger solutions. It is just that it costs more then.

You, 2, mentioned that you saw it as an advantage that it was easy to implement. Did you think more about something else connected to the developer experience? Was it more than just that it seemed simpler than your self-developed solution or?
2: So an advantage that I think of now is that... I kind of never looked into the old database solution. There was 1 who was like the database man. But now we have via, what's it called, the Google Developer Console? So you can just open and look in the

database and like, you can even edit some stuff. Right in the interface. So that's a plus. But that wasn't a reason for us to switch either, it's just the way it is now. It is very user friendly as I understand it. Compared to other databases.
1: Exactly. The biggest reason is just our own time. That we should not have to sit and implement the backend. Also make sure that the database is writable and that the structures we need exist there. All that sort of thing is solved with Firebase. So we don't really have to spend any time at all making sure that we can write to the database and that it can receive the data that we need it to receive. With an SQL database, you have to sit and sort of model the database before it can be used. So that the tables you need exist there. And that the relationships between the tables also exist. But everything like that is solved in Firebase. So you only send in the data that should be there. It's like a huge amount of time saved.

And would you say that is the biggest reason?
1: Exactly. A whole chunk of actual programming time that needs to be put in just goes away with Firebase.
2: Google has put a lot of effort and money and time into perfecting a solution like this. So the likelihood that we could do as well or better is very small. If you don't need special custom stuff maybe. But we're not quite there yet.

1, you mentioned cost too. That previously you paid to maintain a server. But with Firebase you would end up under this threshold that it would cost. Was it also a big factor that you switched immediately? That it would be free in the beginning?
1: Exactly. Our previous solution was incredibly cheap. And that's why we had that solution.
That we were capable of setting up our own server that manages our database. It's perfect. We only need to pay for a server. And that server took care of both having our database and our website and other things. We had a server that was up and running 24/7. But with Firebase, we don't even have to have a server because it's with Firebase. And then we have moved on to other types of solutions to manage the website and so on. So that it doesn't cost any money either. So that was definitely a plus point. We couldn't go straight away from the server solution though. When we switched to Firebase, that wasn't the biggest reason. But we saw that it would be cheaper in the end. When we arranged everything else.

And you thought that it will also be cheaper in the future? Or how did you think about scaling up?
1: Exactly. The upscaling of the old solution had been man-hours that had been spent. So we could have scaled it up by pulling up several new servers in parallel to take

care of traffic or whatever scalability problem we would face. But it's hard to sit and code yourself and probably when we come to such a scenario, when we have to scale it up a lot, we probably need to sit with something other than trying to pull up new servers that will take care of the traffic.
2: Since there are only two of us, we don't really have the time nor the money to have someone who only deals with database management. In order to even keep our heads above water, we have to focus on the product itself or whatever we are dealing with. Develop it forward. The fixed cost of having a database that requires a lot of time becomes very high.

Was support or the community behind the solution something you took into account?
1: It was absolutely a big deal. In case Firebase didn't have a good community, meaning that nothing is written about their solutions or that their documentation is bad. Then it would have been much more difficult to switch to a solution like Firebase. But luckily, Firebase has as good documentation as possible plus they have a lot of users. So if you encounter a problem, you can find solutions to those problems. So it is well used and well documented.
2: We trusted Google as a company when we made the choice too, I guess. Partly that it was Flutter-bound, but also that we have confidence in Google as a company that if they provide a solution like this, it will be good enough for our requirements. It's the brand Google we trust, in part.

Now we get into questions about needs at the database level.
When you switched, what type of data did you have for the most part?
1: We had different ones. For the website, we had made a privilege system to be able to edit the website directly on the website so that you can log in and if you are an admin, you can add and remove posts. At the same time, we saved various types of information that were displayed on the website in our database. After all, it was a pretty clunky system with relationships between privilege levels and information, what should be shown and who can do what. And the same in Flutter, we had built a login. We had a fairly small app that you could upload ads to and there you needed to be able to upload ads and then other users would be able to read but not edit other people's ads. So little of that kind of data about both the user but also their posts.
So User data and posts and other things connected to users?
1: Exactly. It was quite difficult to implement. Then you have to sit there and figure out how the structure should look first before you can even start sending the data to the database. But now with Firebase, all you have to do is submit everything and then you can start testing, and then when you get started and understand how it should

work, you just need to organize the data afterwards and build on it so that you have an optimized solution.
2: It is not uncommon that you change your mind after sitting with it for a week or so.
Perhaps that was what you mentioned earlier as well. That you don't have to sit and model the data in advance. It's faster, if I understand correctly, to get started with something in Firebase?
2: You don't need to take the architect role beforehand. In the same way.
Was that important when you switched or something that you weighed in on?
2: For my part, it was probably more just that I understood it in retrospect and thought "that was smooth".
Okay, so this was not something you looked into beforehand?
2: No, it was more of a reason why we intend to continue staying on Firebase.

1: Something we haven't thought about much is: what is the performance difference between having a SQL and a document based solution? But I find it hard to believe that it would be something that you cannot overcome. That there is a worse look up time on the data or something. In case it is nested or whatever.
Performance, was that something you thought about a lot before switching? Like something you worried about when you switched?
1: No, not really. Based on what I could see on the internet, there were quite large solutions based on Firebase. And it kind of gives confidence that if you get to that point then you might have to put in some time to make it work. Because maybe a slightly slower solution is used. But that is not something impossible. And probably we still save time running on this so that we don't have to sit and optimize solutions that still won't amount to anything.
But was it still something that you looked up a little before?
1: Yes, a little.

What was the process like when choosing your first database?
1: Quite simple. I had experience from previously working at two separate companies using Laravel. So then it was as simple as that being what I knew and felt comfortable with. So it was the easiest thing to get going. I just used the tools I had and knew. That's kind of how I thought.
2: Yes, absolutely. As I said, 1 was responsible for the database. And is. So that's absolutely why we chose it.
1: At the time we were very new, so it felt unnecessary to spend time on things that we won't make any money from. But what works, which we can solve relatively quickly, that's what we used immediately. That solution was much less thought out than switching to Firebase.

So then the most important thing was just to be able to get something that works?
1: Exactly. Have something that can store the data that we need to store when we develop our services.
And what were your needs then?
1: We had some things that needed to store data. We wanted a company website that stored data. Plus another job we've been working on since we started the company. Which would later need to store data online. So there we just needed something temporary, or permanent in case it had turned out that way. But we kind of realized that during the time we've been doing this, that it was a rather problematic solution in many ways. We are too few to have our own server spinning. We want to automate everything. So that we can sit and write code and when we press push, it cascades on all our services. So that they update themselves with the new code that we have written. And such solutions are a bit inconvenient to set up yourself. Now we have a much better structure on such things. We have Hooks on GitHub. So that when we push code, the vast majority of things solve themselves. Database and everything sort of resolves itself. We don't have to sit and tinker with the infrastructure so much.

Appendix D. Interview 2 translated transcript

2023-04-04

Factors

First, we wanted to ask about various factors that you took into account when choosing a database. And we touched on that a bit last time as well, but now we start from the theoretical background of our study and which factors Hassan (2021) believes are important to consider when choosing a DBMS. So we'll go through them and can answer if you took these factors into consideration.

The first factor: Was the type of data you would store a factor in your decision? Type of data both in the sense of what kind of information the data represents and which primitive data types were needed.
1: We used quite a few different primitive data types. And we knew that we would save personal data because we needed a login system. So we took that into account when we looked around for solutions. That's kind of why we didn't want to write our own solution from scratch, but wanted to use some kind of framework or ready-made service. So that security is high and that you don't get, for example, SQL injections.

Did you factor in what amount of data you would store?
2: I spontaneously think no. At no stage so far have we had anything that has been on the verge of being overloaded. We haven't had enough data to really have to care about this yet.
1: There it is only a bonus that what we use is scalable afterwards. But it was and is not a problem at all for us. We don't have an international product with several thousand users.

Schema characteristics? How the data is connected.
1: Exactly, it is important. We wanted there to be an intuitive way to retrieve the data that is linked to the user's data. If they are logged in, it is nice to be able to write good queries to get what they are looking for. Then I think that if you use a framework, you often get a fairly intuitive way of doing it. With Firebase, which we use, it's very easy to create those connections. In SQL, which we used in our previous solution, it is a bit strange how you set up tables and so on. But it kind of works. If Firebase had been worse than SQL, I would probably have backed off a bit.

We talked a bit about the next factor in the previous interview. Cost.
1: Exactly, it's important. Because we want to be as cheap as possible. Both in cost that you have to pay every month, but also time that is spent to make it work. That's

where Firebase ticks all the boxes. It costs us zero because we are so small. We remove as much time as possible in implementation because it works almost immediately with a good API. So that's probably the biggest factor.

And the last factor: Transaction Amount and Transaction Frequency
2: I'd say it's a bit like the Data amount factor. Initially it wasn't that important because we don't have high traffic.
1: Exactly, Firebase is probably better there than our MySQL solution. But we haven't put it to the test really. But we were able to see that Firebase starts charging at 50,000 reads per day. And we are not up to those levels. So if that's their minimum, I think it's pretty clear that their system can handle pretty high loads on their databases.

Is there any factor that you have in mind that we haven't asked about?
2: When it comes to cost. It is not only the cost in money that matters, but it is the cost in time that is the biggest, absolutely most important factor for us.
1: Maybe a little connected to that, ease of use. Something can be very good in all these things. It does not cost anything. It can handle everything. But then maybe the documentation is bad. We kind of checked in general. Is it nice to use?
2: Yes, exactly. How easy it is to solve potential problems.
1: That's kind of how we thought. Can we use this? Is it nice? How much does it cost? Can we afford it? Is it better than what we have today?

Database selection method

Do you know about Roy-Hubara's (2019) method for choosing a database? Or any other method for the database selection process?
1: No.
We would like to go through it, step by step, and then you can say if you followed the step. If you didn't, why didn't you do it? And if you think it would have been good if you had.

Step 1: Gather and specify the data related requirements and express them using a conceptual data model.
1: We did not compile a document with the specifications. What we did was that we had a ready-made solution that worked on our servers. And before we switched to Firebase, we started testing with Firebase and storing the kind of data that we had stored before to see how it works. All types of data that we stored before work with Firebase. So I don't really know if that counts as us doing this step.
2: We are small enough to have the luxury of testing stuff without having to do a lot of research.

And just by testing in slightly different ways, make the decision whether it's good or not. It doesn't have to be huge groundwork for that because we don't have a lot of data.
1: Exactly. We didn't have such a massive migration once we switched.

Did you make a conceptual data model?
1: Not in the switch to Firebase. We usually sketch out how we want to store the data before we start new projects. So it happens on a project basis.
Step 2: Gather and specify the functional requirements that are related to database operations. In other words, what queries should be made.
1: Yes, we did that but informally. Because we had a working previous solution and then tested that everything that worked there also worked with Firebase. Then we also checked the documentation of the Firebase API and saw that everything we will ever need can be done with Firebase.

Step 3: Gather and specify the non-functional requirements that are related to the data requirements and queries.
1: Absolutely, but also informally. We took a closer look at Firebase, for example on their security. They use server-side encryption and many standard things that you should have in your databases to make them secure.

Then we can ask as a summary question of steps 1, 2, 3. Do you think it would have helped you to do these steps formally?
2: Yes, it probably would have. If you made it very clear what the choices there were. Especially if we had a few more options on the table. We kind of found Firebase and just checked that it could do what we needed. And we trust Google as a brand. So that convinced us to use it right away. But if you are going to compare several different solutions, it is probably very helpful to have it in print and go through it a little more formally.
1: Exactly. However, I think for us as such a small company. After all, it will be a cost in hours to make such a comparison. And how important is it for us to find the absolute best? It's not that important. As long as we find something that is good enough, it is much better for us to focus on developing our products. To make the comparison a bit informal in our scenario, I see it as better because it goes faster. You avoid the absolutely biggest traps. Even if it won't be optimal, it will be good enough.
2: Yes, for a large company, perhaps a difference of 0.1% in efficiency can save several hundreds of thousands (SEK). But for us it would be maybe ten (SEK).

Step 4: Based on the above, the method considers dividing the conceptual data model into fragments, each of which has different characterizations (such as

different access frequency, different performance requirements, and different consistency requirements).
1: So far there has been no reason for us to have multiple fragments. It increases the complexity as well and we probably have nothing to gain from it. We're so small, and Firebase also scales automatically.

Step 5: Select the most suitable database model/system for each fragment. This will be based on a general-purpose pre-defined profile for each database model. A predefined profile consists of a set of non-functional properties associated with each database model.
1: So when you have all this fully documented and specified on paper, you would be able to match them with different solutions?
Yes. Both that you could have different database models or structures in one and the same database manager, but also that you could have separate, but connected, systems. But since you didn't split into fragments, step 5 would just be about choosing a system/model.
1: We did not make a comparison between several different alternatives. We only compared the old solution against Firebase.
2: It was perhaps a bit naive actually to only look at Firebase. But so far so good.

Step 6: Design the selected database model/system for the different fragments.
1: That we did!

If you look at the method in its entirety. Is there anything that you think is good or bad in your case, considering the size of your company?
1: It is absolutely a sensible way to do it, to arrive at a system that makes sense. I would guess that if you use this method, you would arrive at a system that always works. It is general.
2: I think there is a lot to do. I would guess that this is designed for large companies. The larger the company, the more applicable this method is.

So perhaps this method could be thinned out a bit to make it more suitable for a small company?
2: Yes I absolutely think so.
1: Exactly. It is great to have a formal structure of what is important to consider when choosing your database. So a step by step description of what to look at when making the choice. It makes sense.
2: I agree. Because this really is a comprehensive guide. Which should work for all companies. And then if you come to the conclusion as a company that "we can ignore

this part". Then you do it. But you can still have research at the bottom and know that you won't miss anything in a process like this.
1: The only parts of this approach that we didn't have in our process is that we didn't document the requirements and we didn't even think about fragmentation. But steps 1, 2 and 3 were absolutely included, especially 3. That was the most important thing. If you factor in cost as a non-functional requirement. Steps 1 and 2 were kind of obvious to us. That it would work with basically any solution. And then step 3 was the only thing we used to make our decision.

Characteristics

Is your planning carried out mostly face-to-face, informally and unstructured?
1: We try to have one meeting per week where we go over what we are doing, but these can quite often get mixed up in talk during lunch. So about once a month we try to have a slightly longer meeting to check where we are going. So far we have had goals for the year which are like financial and project goals. Our meetings are informal, we have a meeting protocol that is rather poorly updated. The monthly meetings are also quite informal, but there we usually do some counting and documenting. And so far we've written down what goals we have and we have checked how we've performed against them at the end of the year. But all meetings are basically face-to-face, we try, even if it is informal, to keep a structure. So we know what to do going forward, that means we try to know what to work with for at least the coming six months/year.

Laporte et al. (2008) say that small companies tend to have Human capital centered competitive advantage while larger companies have Organizational capital centered competitive advantage. Is it true that you have Human capital centered?
1: I think that's right, we work with what we have competence. We don't choose to work with things we don't know well. So our entire workforce has competence in what we're dealing with. I don't know if I'm interpreting it correctly, but I assume that larger companies have a bit more diversified knowledge, which means that we can sometimes have an advantage on specific things.
I have interpreted this characteristic difference to mean that small companies have the competence of the employees as their main resource, while larger companies have a lot of value in the way they are organized.
1: Ah yes, we do our best but that is our main problem. No finance or HR department to manage things.

Would you say you are more flexible because of your size?

2: Yes. Because we have no one to stop us when we make decisions and no bureaucracy. We basically have a non-existent bureaucracy. A pretty strong argument that we are very flexible.
1: I would just add however, we don't have the muscle, meaning the money muscle to be able to choose all the options available. But still, it's flexible.
2: If a super high-tech solution appears tomorrow that solves all the problems and then some. Then it takes us a week to start using it.

Would you say that this flexibility has some kind of down side?
2: It is not the flexibility itself that has a downside. It's probably more about being a small business that has downsides. And one of the upsides is flexibility.

Do you feel a lack of organizational structure?
1: Yes, we cannot focus on development alone. We need to do the accounting, answer all emails, and stuff like that. And another thing. It's good in that way that we can do VR one week and then we can do AI the next week. And then we can sit with websites in the third week. But we have no chance to sit with VR and make Assassin's Creed. Although we have great flexibility in what we can do, the amount of output is limited by the fact that we are few.
2: We need to make smart projects instead of Brute Force producing a huge project.

Are there more strengths or positive aspects which come as a result of you being a small business?
1: We have the upside that we can take on many different types of projects. However, I wonder if it is because we are a small company, or if it is just our skills. But when I worked at a larger company and had an idea I wanted to do, it took several months before it could be approved. Such things do not happen to us. We just ask each other if it sounds like a good idea and off we go.
2: Yes, on new projects or to sort of pivot old projects, the starting distance is very short
or a very low threshold to pivot anything.
1: Another upside is that we can be at the cutting edge with new solutions. If you hire us, you can sort of get access to what came out last week. It may not be as stable, but you sort of access new technology in a completely different way because of our size. Whereas if you hire a large company, they often make stable and old solutions. We have the strength to be able to provide what is brand new all the time.

Now to the negative aspects. Are you vulnerable if a single project goes badly? Is the company's finances largely affected?

2: Yes, absolutely. It has a big impact if a project ends or a customer, for example, will not return. That has the biggest impact financially.
And do you see it as something that is due to your size?
2: Yes, it will be much more fluctuating. So very uneven income streams compared to larger companies. For them, there will be a much more even distribution of work over time.

Would you say you have a low budget for each individual project?
2: Yes, definitely. It is basically a few thousand (SEK) per project. We have no investors. We have no risk capital in this company either. Everything is our own invested capital and income from direct services that we only have from our competence.

Having a small budget for projects often leads to limited resources for, among other things, quality assurance. Is that true for you?
1: It is about what the customer wants. If they want something quickly, we won't have time to do much quality assurance. But if they pay money for the project and want something good. Then we make that quality.
2: It will be different in how to reason here depending on whether it is our consulting activities for other companies or whether it is our own products that we develop. I think we've talked about it quite a bit. This quality assurance activities may also include fine-tuning the products before releasing them. We are very good and quick at producing a proof of concept when we have an idea and then we just go and it will be okay. But the final fixes for a project often take a brutally long time compared to the core functionality itself. We don't have time to bug test everything and we don't have time to completely ensure high quality in all aspects.
1: That is true. On our own things, we definitely keep a lower standard than larger companies with more budget. We can't sit and test everything we've built. Actually if you look at it it's probably 30% development and 70% quality assurance. So if it takes us three months to develop something, we should sit for ten or seven months and test it. But we can't afford that.

Corrective Post Delivery Maintenance?
2: I would say so.
1: It's not like we're leaving anything to rot. But it could definitely be better.
2: Since we have a low budget, we can't afford to hire someone to do it while we do other things. According to our own priority system, we have things that are much more important to solve than small bug fixes and polishes on old projects.

Does a small budget lead to limited resources for training and skill development?
1: Yes, we don't have traditional training.
2: I would say that I learn more being in this company than if I were to be in a larger company. But absolutely, we don't sit in courses and pay for someone to explain things to us.