A comparative analysis of ORM frameworks in .NET and Spring Boot

Bachelor of Science Code & Context

Frederik Wulf
Matrikel-Nr.: 11148147
Wrombacher Str. 14
57392 Schmallenberg

First reviewer:	Prof. Dr. Stefan Bente
Second reviewer:	Prof. Dr. Frank Schimmel

Köln, TT.MM.2023

Erklärung

Ich versichere, die von mir vorgelegte Arbeit selbstständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer oder der Verfasserin/des Verfassers selbst entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben. Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

Anmerkung: In einigen Studiengängen steht die Erklärung am Ende des Textes.

Ort, Datum

Rechtsverbindliche Unterschrift

Abstract

This analysis aims to compare and contrast the strengths and weaknesses of two commonly used Object-Relational Mapping (ORM) frameworks, Entity Framework (EF) Core for .NET and Hibernate for Spring Boot. The objective was to identify the relative benefits and potential drawbacks of each ORM to provide developers with a clearer understanding of their capabilities. To facilitate this comparison, two projects were developed for each framework. The findings indicate that both EF Core and Hibernate offer a variety of useful features, however with different implementation experiences. EF Core needs more extensive configuration and has a more complex query logic, which increases implementation complexity, provides added flexibility and is provided by comprehensive documentation. In contrast, Hibernate for Spring Boot has a more minimalistic approach, requiring less configuration and enabling more rapid development. However, its documentation is less extensive and comprehensive compared to .NET's EF Core. The insights from this study can help developers to compare the frameworks in terms of ORM implementation.

Keywords: Object Relation Mapping, .NET, Spring, EF Core, Hibernate, C-Sharp, Java

Contents

1 Introduction

In the current digital world, a lot of applications are developed and hosted on the web, processing a large amount of data. This data is mostly persisted in databases. The usually used Relational Database Management System (RDBMS) employs patterns that differ significantly from the utilized Object Oriented Programming (OOP) approach for these applications, thereby implementing a proper connection to a database is a complex task. As Vial (2018:p.45) states 'ORM engines are by no means a requirement for interacting with a relational database. Developers can use low-level components to manage their object model's persistence layer (as was usually the case in the past). In those instances, developers manage the generation of SQL queries and the recreation of objects from SQL result sets while sending queries to the database directly.' and Vial (2018:p.49) also states 'ORM engines save developers time and frustration by removing some of the grunt work associated with object persistence. Nonetheless, it is important to be mindful of the characteristics of these engines to take full advantage of their potential.'

This analysis compares two major frameworks and their respective ORMs. Firstly, the long-standing Spring Boot framework. Spring uses the statement 'most [of our] services today are all based on Spring Boot. I think the most important thing is that [Spring] has just been very well maintained over the years...that is important for us for the long term because we don't want to be switching to a new framework every two years.(Paul Bakker, Senior Software Engineer, Netflix)' (VMware, 2023*c*) to describe itself. Secondly, a large competitor, Microsoft's .NET. .NET which cross-platform capabilities were only released in 2016 gained more popularity since. For the ORM comparison the most widely used ORMs for each framework are analyzed. For .NET this is EF Core and for Spring this is Hibernate. There are a lot of different topics that can be compared this study will take a deeper look at the ecosystem around these ORM's and also the basic implementation and functionality in these frameworks.

2 Goal

The central objective of this analysis is to provide a analysis two distinct frameworks, .NET and Spring Boot. The analysis will cover the implementation of ORM's in their ecosystems. Especially outlining their respective strengths and weaknesses in the context of a new and small-scale project. This exploration involving a comparative analysis of the comprehensiveness, clarity and usability of the documentation associated with each ORM. Further, a broader overview of community support for both frameworks is conducted, offering insights into their popularity in their respective community. Following a feature analysis, aspects such as configuration options, entity mapping capabilities, query building and handling, and migration creation are observed. Specially the ease of use is covered. This detailed comparison is should provide developers and other potential stakeholders with a more robust understanding of these frameworks and their respective ORM implementation and can contribute to an informed decision making of choosing between these frameworks.

This analysis is conducted with two developed project with the same scope and requirements, which serve as a case study. One project for each framework. The project is an offer service in a larger ecommerce microservice ecosystem. Within chapter 5 a broader overview of the project is given.

3 Basics

This chapter aims to provide a basic introduction to ORMs, its problem it wants to solve and the ecosystem of the project scope. It will serve as a fundamental understanding of ORMs, their purpose and why they are important. Throughout this chapter we will dive into the Impedance Missmatch Problem, exploring where OO Language models and relational data models differ. Further we take a look at how ORMs try to tackle these differences. In addition, the .NET and Spring ecosystem are examined, as well as the function that ORMs have within both. Further this chapter covers a deeper look into relational databases and why they are so impotent in the software development landscape. Finally, this also covers the technologies utilized in the projects.

3.1 Impedance Missmatch Problem

The impedance missmatch refers to the missmatch between RDBMS data models and OO data models. Typically, RDBMS models adopt a table format, consisting of rows and fields. On the other hand, OO data models are built using classes and objects. An object is an instance of a class, which consists of properties and methods. These basic structures already are fundamentally different. In the Hibernate documentation JBoss (2023*b*) breaks down what problems occur when trying to integrate these two paradigms together:

3.1.1 Granularity

The two models can have a different level of detail, which can lead in the granularity problem. This implies that for example the OO model can be more granular than the relational data model, for instance, when the number of classes in the OO model exceeds the number of tables persisted in the database. (JBoss, 2023*b*)

3.1.2 Inheritance

Inheritance is a fundamental concept in OO. It allows one class to inherit the characteristics (properties and methods) of another class. It enables the creation of a hierarchy or some kind of relationship between classes, where one class can acquire and extend the features of another class. There is nothing similar defined in any RDBMS. Therefore inheritance may be a problem when using OO and RDBMS together. (JBoss, 2023*b*)

3.1.3 Identity

Both OO and RDBMS use different paradigms to check for equality. In a RDBMS the sameness is ensured through a unique primary key. If OO languages, for example such as Java and C#, are observed, they have a different approach. First there is the equals operator (`a==b`), which will check wether both variables have the same memory address. Then there also is the `equals()` method,where a developer can implement their own definition of equality. (JBoss, 2023*b*)

3.1.4 Associations

Associations between objects are also handled differently in both models. In OO languages, these are usually represented by unidirectional or bidirectional references, one object referencing another allows for interactions between them. The referred object, however, does not need to hold a reference to the original object. While on the other hand, RDBMS rely on the concept of foreign keys. A foreign key is a field in one table that refers to the primary key in another table, establishing a connection between them. (JBoss, 2023*b*)

3.1.5 Data navigation

Another issue is the way of data accessing and navigation because OO navigation differs significantly from RDBMS navigation. OO Language structures data with associations between objects like explained previously, with these associations it is possible to navigate from one association to another. This can also be described as a form of a network or graph-like structure. Meanwhile the associations in RDBMS defined with foreign keys are combined using the SQL `JOIN` operation. It will combine related tables and retrieve the desired data in a single query, into a single table.

3.1.6 Unsupported or Non-Simple Types

Mapping becomes more complicated when the class attribute type to be mapped is not a simple (primitives) type or a type not supported by the relational database. For example, most relational databases do not support boolean types, which are common class property types. (Chen et al., 2020:p.3)

3.2 Object Relational Mapping (ORM)

Object Relational Mapping tries to solve the impedance missmatch problem. As discovered in the previos section the paradigms used by OOP and RDBMS data models diverge considerably. To overcome this missmatch an ORM can be used as a solution. It resolves the discrepancy between the OOP and RDBMS data models, enabling developers to operate with a RDBMS while working with OOP principles and concepts. In other words it allows the developer to work with objects and classes while the ORM abstracts the database and handles the conversion of tables, rows and columns into objects and class.

In the usual case an ORM maps a class to a table, an object to the rows and the properties to the columns.

OO Concept	**RDBMS Concept**
Class	Table
Object	Rows
Properties	Columns

Table 1 Object and relation mapping, adapted from (Chen et al., 2020:p.2)

In some cases a simple mapping is not possible, due to complex data types, when a property needs to be understood as a combination of columns or for relationships between different objects. For such a case a model or configuration is necessary which will dictate how these different classes or properties can be combined. With this solution, ORMs already solves the granularity and association problem. Further the unsupported or non-simple types can be resolved thought abstraction. ORMs abstract these into simple types for example a boolean type property is abstracted into a single character (usually 0 or 1) (Chen et al., 2020:p.3).

The inheritance problem is a bit more complex and brings multiple strategies to solve it. Microsoft (2023*b*) shows different strategies how to solve these problem, Table per Hierarchy (TPH), Table per Type (TPT) and Table per Concrete-Type (TPC). TPH is an inheritance strategy where all classes in an inheritance hierarchy are mapped to a single database table. The table contains columns for all the attributes of the classes in the hierarchy, with some columns allowing NULL values for attributes that are not assignable to certain subclasses. A discriminator column is used to differentiate between different subclasses. TPT is an inheritance strategy where each class in the inheritance hierarchy is mapped to a separate database table. The table representing the base class contains common attributes, and each subclass table contains the specific attributes. Relationships between tables are typically established through foreign key constraints. TPC, also known as Table-per-Concrete-Class, it maps each class in the inheritance hierarchy to a separate database table. Unlike TPT, there is no base class table. Each table contains both the common and specific attributes. These strategies allow for different trade offs between table structure, performance, and querying capabilities. The choice of strategy depends on the specific requirements and considerations of the application.

To sum everything up, ORMs bridge the gap between OOP and RDBMS. ORMs handle the mapping between objects and database tables, with different mechanisms, tools and patterns to solve the different problems.

3.3 .NET

The .NET ecosystem, as defined by Microsoft, is a cross-platform, open-source framework that provides a versatile environment for the development of many different types of software, including desktop, web, microservices, machine learning, and cloud computing applications, among others.

.NET offers a variety of programming languages, including C#, F# and Visual Basic. It supports OOP and Functional Programming paradigms, making it a flexible choice for developing different kinds of applications.

In 2016, Microsoft introduced .NET Core, a modular, open-source, and cross-platform version of .NET, to modernize the framework and allow it to run across different operating systems. As of May 2023, released the eighth version.

The .NET ecosystem is well supported with a rich suite of development tools by Microsoft. It also has a strong community that contributes to a vast collection of libraries and frameworks, enhancing its functionalities further.

The .NET ecosystem is supported and furhter developed by Microsoft. They say its

architecture and comprehensive collection of libraries simplify complex programming tasks, allowing developers to focus more on writing business logic rather than dealing with low-level programming details. (Microsoft, 2023*d*)

3.3.1 ADO.NET

The main data access method used by the.NET framework is ADO.NET. It offers a collection of classes and APIs for utilising access to many kinds of databases. ADO.NET offers the `SqlConnection` and `SqlCommand` classes for creating database connections and running SQL commands. Compared to higher level ORM frameworks, it offers a lower level of control over database operations and lets developers work with raw SQL queries. (Microsoft, 2023*a*)

3.4 Spring

The Spring framework ecosystem is a framework for the Java platform. It was first released by Rod Johnson in 2003 and is now maintained by Pivotal Software, which is a division of VMware. The broader Spring ecosystem includes other specialized projects built on top of the Spring Framework, such as Spring Boot, Spring Data, and Spring Cloud. Each of these provides solutions for different types of software.

Spring was primarily designed to simplify enterprise application development. It achieves this with a focus on speed, simplicity, and productivity. Its modular design allows developers to choose only those components necessary for their applications, which helps to avoid unnecessary dependencies.

The Spring Framework is a Java based framework. However, it also provides class support for other JVM languages like Kotlin, Groovy or Scala, which means these languages can also be used to write Spring applications. The Spring community is active, with numerous resources for learning and troubleshooting. (VMware, 2023*b*)

3.4.1 Java Database Connectivity (JDBC)

Java Database Connectivity is a Java API that provides a low level, way to interact with relational databases. It is comparable to ADO.NET from the .NET ecosystem. It enables programmers to create connections, send SQL commands, and handle the results. JDBC offers a collection of interfaces and classes that abstract the underlying database processes, allowing programmers to write SQL queries directly into database-specific code. SQL statements, result sets, and database transactions must all be handled manually. (Oracle, 2023)

3.4.2 Java Persistence API (JPA)

Java Persistence API is a Java specification that provides a higher level of abstraction and rules over JDBC. It's an ORM framework that enables programmers to translate Java objects into relational database tables and carry out database operations utilising OO patterns. Entities are Java classes that represent database tables, and JPA introduces the idea of entities and provides annotations and APIs to define the mappings between

entities and database tables. Hibernate is based on JPA and provides an additional layer of abstraction as we will discover later on. (IBM, 2023)

3.5 Relational Database (RDBMS)

The relational database model is based on the idea of representing data as sets of tables, also known as relations, where each table consists of rows and columns.

One fundamental aspect of relational database systems are the set of data integrity rules. These rules include entity integrity, which ensures that each row in a table is uniquely identifiable, and referential integrity, which guarantees that relationships between tables are maintained through key constraints.

Another key concept in relational database systems is the use of a structured query language (SQL) for data manipulation and retrieval. SQL is a specification, which provides a standardized way to interact with the database, allowing users to execute CRUD operations. This enables users to perform complex queries and join tables.

Relational database systems also support ACID properties, which stand for Atomicity, Consistency, Isolation, and Durability. These properties ensure that database transactions are executed reliably and maintain data integrity. Atomicity ensures that a transaction is treated as a single unit of work, either fully completed or rolled back in case of failure. Consistency guarantees that the database remains in a valid state before and after a transaction. Isolation ensures that concurrent transactions do not interfere with each other, and durability ensures that committed changes are permanent and survive system failures. (E.F. Codd, 1970)

It is crucial to compare various popular DBMS types and understand why RDBMS dominate the market. It may seem intuitive to choose a document or object-based databases due to the similarity of their data model with OO data models. While this similarity simplifies the data modeling process, the popularity and demand for these databases do not mirror this potential advantage. RDBMS owe their dominance to several key advantages over other DBMS types. The structured nature of the RDBMS data, organized into tables, rows, and columns, facilitates simplified querying and manipulation. This structure also enhances storage efficiency and reduces data redundancy, as recurring data only needs to be stored once. Data integrity, a crucial aspect of any database system, is another strength of RDBMS. These systems provide mechanisms to uphold referential integrity. Multi-user accessibility also is a fundamental requirement. RDBMS handle this need through locking mechanisms, which prevent concurrent data modification by multiple users, thus maintaining data consistency(Mandapuram and Hosen, 2018). Despite these benefits, it is important to note that alternative database systems may present advantages when faced with specific application requirements.

This observation is supported by the little diversity of DBMS types among the top ten used database engines. As an illustration, within these top ranked systems, only three NoSQL databases, MongoDB, Redis, and Elasticsearch, are included.

Given the research above, it is not surprising to observe that most ORM tools specialize in supporting RDBMS. RDBMS's structured data organization, robust data integrity mechanisms and efficient multi-user handling capabilities make them a practical choice for

Ranking	DBMS	Database model	db-engine score
1	Oracle	relational	1256.01
2	MySql	relational	1150.035
3	Microsoft SQL Server	relational	921.60
4	PostgreSQL	relational	617.83
5	MongoDB	document	435.49
6	Redis	Key-value	163.76
7	IBM Db2	relational	139.81
8	Elasticsearch	Search-engine	139.59
9	Microsoft Access	relational	130.72
10	SqlLite	relational	130.20

Table 2 Adapted from (db-engines, 2023)

a wide array of applications. In order to make database interactions for developers easier and more efficient, ORM technologies naturally lean towards supporting these widely used systems.

3.5.1 Migrations

The process of making updates or modifications to the database schema is called a schema migration or just migration. These adjustments can be of different kinds, adding a new column to an existing table or altering the data type of a column, modifying foreign keys, or even dividing or merging tables. Schema migrations frequently happen within enterprise software updates or releases.

Usually a migration of a database is performed by at least one but usually multiple SQL scripts. For a single change or a small amount of schema modifications this works nicely. Looking at large scale enterprise software, schema's are usually already quite complex and are changing regularly. Thus the SQL scripts can become more error-prone (Marks and Sterritt, 2013).

Therefore, modern systems usually have some migration and schema version management systems. A Schema migration process of a database or data storage system should always run without losing or corrupting existing data. Some systems can ensure that the database structure remains in sync with the evolving needs of the application or system. By employing migration version management systems, developers can automate and simplify the process, making it easier to track and manage changes to the database schema

over time. These systems often provide tools and utilities to generate migration scripts, apply them in a controlled way and keep a record of applied migrations, enabling efficient collaboration among developers and ensuring the integrity and consistency of the data and the schema.

3.6 Technologies used

The selection of the technology stack for the developed project dependent on two essential considerations. The first aspect was the familiarity with certain technologies, ensuring quick feedback loops and efficient progression. The second factor resolved around overall usage or download numbers of the technologies. Some of the chosen technologies are exchangeable as long as it stays in the .NET or Spring ecosystem.

3.6.1 MySQL

MySQL is an open-source RDBMS that is supported by Oracle. It is widely used in the industry due to its popularity and extensive features. MySQL offers various functionalities such as stored procedures, triggers, views, and full-text search. For this study, MySQL can be seamlessly replaced with any other RDBMS, and such a change would not have a significant impact on the project.

3.6.2 ASP.NET Core

ASP.NET Core is a web development framework by Microsoft that enables developers to create dynamic web-applications within the general .NET ecosystem. It supports programming languages like C# and Visual Basic. With features like MVC architecture, built-in authentication and data controls, ASP.NET simplifies web development and offers flexibility to run applications on a wide range of platforms. (Microsoft, 2022*b*)

3.6.3 Entity Framework (EF) Core

Entity Framework Core simplifies the process for developers to access data stored in a RDBMS. It is an open-source tool, it's developed and supported by Microsoft. EF Core employs Entity Data Modeling, where an entity class corresponds to a table and properties to columns in an RDBMS. EF Core supports various RDBMS, including all RDBMS of the top ten previously mentioned, but it does not support any NoSQL database systems. Compared to alternatives like NHibernate or Dapper, EF Core is the most widely used ORM in the .NET ecosystem (nuget.org, 2023). To query data, EF Core uses Language-Integrated Query (LINQ), a strongly typed query language. LINQ queries are written against the defined (entity) class. EF Core translates these LINQ queries into SQL queries and executes them at the database level. EF Core also features a built in migration system. This system enables model changes to be converted into schema changes. It can accommodate all kinds of changes, such as adding, deleting, or modifying tables, columns and rows. While simple migrations can be achieved through built in methods, developers can also write an SQL script for more complex changes. Most changes for the schema can be automatically generated with a CLI command and then be applied to a database. This

capability allows for smooth database evolution as the application's object model is further developed. In addition to these features, EF Core provides support for change tracking, transactions, caching, and an in-memory database for easier testing capabilities (Microsoft, 2022*a*).

3.6.4 Spring Boot

Spring Boot is the counterpart of ASP.NET Core just in the Spring ecosystem. Like ASP.NET Core, it provides features for creating web applications. It is developed and supported by VMware. With Java, Kotlin, Groovy and Scala it also supports wide range of different languages. It provides defaults and automates configuration tasks that are common to most projects, allowing developers to focus on the unique parts of their applications. Being a Java framework, Spring Boot can run on any common platform that supports the Java Virtual Machine (VMware, 2023*a*),

3.6.5 Hibernate

In August 2023 Hibernate is the most popular implementation of the JPA specification in the mvnrepository (maven repsoitory). It is a ORM framework that provides additional features and capabilities beyond the standard JPA specification. Hibernate simplifies the process of database access by automatically generating SQL statements based on the defined mappings and performing CRUD operations on entities. It also offers caching mechanisms, lazy loading, and various optimization techniques to improve performance. Hibernate abstracts away the underlying JDBC operations and provides an easier and more intuitive way to work with databases in the Java or Spring ecosystem. Similar to EF Core, Hibernate supports all of the RDBMS in the top ten, with the exception of Microsoft Access, but it also has expansions for connection to other types of databases like Hibernate OGM (Object/Grid Mapping) for Redis or MongoDB. (JBoss, 2023*a*)

3.6.6 Flyway

Flyway is an open-source tool used for database migration in the Java and Spring Ecosystem. It is developed by Redgate, which claims that Flyway has been the industry standard for over 20 years. Trusted by large companies such as Google, IBM or Pfizer, Flyway simplifies the process of writing and running migrations by allowing the addition of SQL scripts to a designated migration folder.

In addition to migration capabilities, Flyway offers different other features. It can repair the schema_history table in case of issues. The `schema_history` table is important for Flyway, since it keeps track of the current version of the database schema. When working with an existing database, the `baseline` command can be used to create an initial schema migration based on an existing database. The `validate` command checks if all the created scripts have been applied to the database it is executed against. Lastly, the `clean` command is available to drop the database, which can be helpful for testing or starting fresh in development.

Flyway provides three different plans: a free community plan, a paid Enterprise plan, and a paid Teams plan. The paid versions in particularly, offer numerous additional fea-

tures, especially in terms of DevOps functionalities. All versions of Flyway can be utilized via the command-line interface (CLI) as well as through a graphical user interface (GUI). (Redgate, 2023)

4 Procedure for the analysis

In this study, two ORM frameworks within distinct ecosystems, .NET and Spring Boot, are compared in a mostly qualitative way. To ensure a fair comparison, two applications with identical functionality were developed in each ecosystem. To ensure sameness in functionality each framework had a similar test suite. The next section goes into the application's development steps, providing an overview of how the projects were structured, developed and implemented. However this section goes into the analyse and comparison principles.

The comparison begins with a exploration of the broader ecosystem and the community of each of the main frameworks. This consists of an analysis of the size and popularity of the ASP.NET and Spring Boot ecosystems, as well as a research of the documentations provided for the frameworks in compliance with ISO 26514 standards for documentation design. ISO/IEC prescribes that documentations should contain a system design descriptions, user guides, installation guides and reference manuals. Due to the amount of documentation in both ecosystem this analysis will only be done with samples from the official documentation pages, which were best case also relevant for the sample applications.

For the community research, different tools and websites for software development are used. The GitHub Stars and Forks are used as one indicator for collaboration, liking and usage of the different tools. Further the total number of questions posted on Stackoverflow for each ecosystem are analysed, since the active use and the amount of solutions for different problems can be derived from this. Various relevant tags associated with these ecosystems are analyzed. Additionally the Stackoverflow Developer Survey, as a highly regarded and answered survey for developers, is evaluated. The survey covers a wide range of different topics to the entire development ecosystem, but for this analysis, a selection of questions and topics that are relevant to this study is made.

After the broader look of the frameworks, the ORMs features are further examined. The two frameworks offer various types of functions and implementations for the same functionalities. For the feature comparison special core functionalities are selected and divided into four different sub topics, Configuration, Entity Mapping, Query Building and Migrations.

In the configuration aspect, Spring Boot claims for itself, that it 'Provides opinionated starter dependencies to simplify your build configuration' (VMware, 2023*b*) and 'Automatically configure Spring and 3rd party libraries whenever possible' (VMware, 2023*b*). Spring Boot is confronted with the fact that it says that it should be easy to setup and use. The configuration needed to setting up an ORM, in this case Hibernate is contrasted with the implementation of EF Core in an ASP.NET application. It will be examined what still needs to be configured and how much code needs to be written for this.

Subsequently the mapping of the OO models to a database is explored. The resultant database models, and the effort required (in terms of time and code) in both Spring Boot and .NET will be evaluated.

Further the querying of data in the database is analyzed. The basic section of EF Core observed that EF Core makes use of LINQ to write the queries. This is a different approach to the Hibernate's or JPA's Derived Query Methods and Hibernate Query Language (HQL). Both abstract the usually in RDBMS used SQL away from the developer. Both have different approaches to write queries, both approaches will be analyzed in terms of lines of code needed to write and also what time is required to set up the same set of queries.

The last comparison is the creation and management of migrations, which is referring to the observation in the basic section a common task in application development. The .NET's ORM EF Core already brings some features for migrations. Unlike EF Core, JPA and Hibernate lack built in migration features, necessitating the use of a third-party tool. To compare the the capabilities of migrations two projects steps are implemented. A first step will only evaluate the initial setup of a database for the fist time without a migration tool, at least for the Spring Boot application. Since for basic setup hibernate has some capabilities. In a second step a migration will be applied, where the existing table and data structure is modified. Throughout this investigation, the required configuration, the different features for managing migrations and the required effort of time and code are analyzed.

To measure the lines of code and time the development steps where divided and documented. Usually a step was also a commit. For measuring the total time for a step simply the start time and end time was taken. To measure the lines of code, for every step the, total lines where measured with the JetBrain IDE pattern search. The pattern `^(?!.*[{}]s*$).*\S` which removes empty lines and also removes indentation style. Specially the remove of the indentation was important since in Spring Boot code the K&R style is used, which compared to the Allman style, regularly used in .NET, prodcues less code just for indentation. So with that pattern any line just for indentation is removed.

For a further comparison the ISO/IEC 25010 standard for software and data quality is employed to the previously descried features. ISO/IEC 25010 defines a model for assessing and specifying the quality characteristics of software products. It provides a structured framework to assess software quality by breaking it down into eight main quality characteristics. Not all characteristics defined by the ISO are relevant for this analysis. Therefore a special focus is given to the metrics of Functional Suitability, Usability, Reliability, Maintainability, and Portability, as these aspects in particular lead to the broader goal of the analysis. Functional Suitability will be assessed by evaluating how well each ORM handles the mapping of entities and construction of queries. This evaluation aims to detect what types of mappings and tasks each ORM can handle. Usability will be measured by comparing the frameworks in terms of configuration, entity mapping, query building, and the creation of migrations. These tasks are performed regularly, therefore the ease of use, speed of learning, and accessibility of each ORM are of signified importance. Reliability is particularly crucial in the context of migrations. A highly reliable ORM should exhibit strong fault tolerance and offer effective recovery mechanisms. Maintainability will be assessed based on modularity, reusability, and testability, particularly in the context of query building. Lastly, Portability will be examined by assessing the steps required to configure

each ORM. The aim is to understand the ease or difficulty of setting up each framework across different environments.(ISO/IEC, 2011)

5 Project integration with ASP.NET and Spring Boot

The following chapter shows the implementation and integration of each framework. The subsection of prerequriments will go into steps and application design before actual development. After that each framework has its section where the development process is explained. This will follow the approximate sequence of development and will also explain some sample code snippets.

5.1 Project and Model Structure

5.1.1 Prerequirements

The project developed for the analysis is part of a larger system. The complete system is an E-Commerce microservice project developed in advance of this analysis [1]. The application makes use of the Domain Driven Design (DDD) approach. This design offers an approach for creating an OO-Model for a large system. The already developed project consisted of four different microservices developed with ASP.NET, Apache Kafka and PostgreSQL. The application is able to handle account and user management, inventory, shopping-cart and fulfillment processes. The products offered to the customer where simultaneously the products that are in the inventory. In the context of the reprocessing and documentation of the C-commerce project, is already mentioned that an offering service could be a valuable extension for the further development of the application (Frederik Wulf, 2023). Such an addition would enable the application to effectively manage the products being offered. Currently, the system automatically offers all products in stock without any form of administration.

In preparation of the actual development, a set of functional requirements was formulated. These requirements should provide a clear understanding of what should be implemented. During the formulation process of requirements, some aspects and topics, specially for this analysis, where considered. These aspects developed from the problems ORMs want to solve like mapping, inheritance and migration where considered. From this considerations the following requirements where defined:

1. A single product should be offered for a defined price
2. Multiple products can be offered together as a package for a price
3. An offer may expire
4. An offer can be reduced for a defined time
5. An offer can be created for a specific marked / country

The requirements one to four are developed in a first development step. Later in the project phase the fifths requirement the Localization of offers is set up, to be able to migrate the projects database. In the next section these requirements are translated into some object models.

[1] https://github.com/fredyyy998/ecommerce [July 2023]

The project was developed with a GitHub repository [2].

5.1.2 Object Model

With the initial set of functional requirements the following object model is designed.

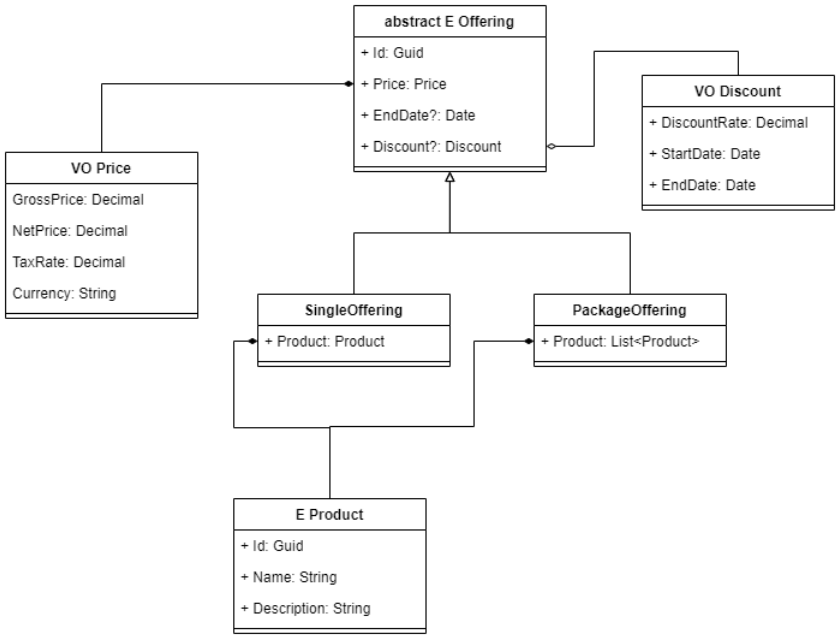


Figure 1 OO Model or Class Diagram of Offer Service step 1

It incorporates different concepts. First with the single and package product offer requirement, a hierarchy is designed. This focuses on the inheritance problem, mentioned in the basic section. This ensures an inheritance is implemented and the different solutions can be analyzed.

Within the model, different entities where designed, the already mentioned `Offer` and also a `Product`. These two entities create another scenario, the associations or references which are also a problem already mentioned in the basics section. The `SingleOffer` has a many-to-one relation to the `Product` entity and the `PackageOffer` has a many-to-many relationship. This scenario will show how the two ORMs are resolving such references between classes.

For the last two objects `Price` and `Discount` the concept of value objects, occurring of DDD, are employed. These objects will not have an identity.

As already mentioned a Localization requirement is added to simulate a migration. With the new requirement a further developed object model is created.

This requirement is mainly designed in a new entity called `Localization`. This entity has several properties, including the `Country Code` which uses the alpha-2 country codes defined in ISO 3166 [3]. This property also serves as identifier. Additionally, the entity owns the `CountryName`, which also defined within the ISO, and further extends to incorporate the country name translated into the local language of the respective country.

[2] https://github.com/fredyyy998/ecommerce/tree/thesis
[3] https://www.iso.org/iso-3166-country-codes.html

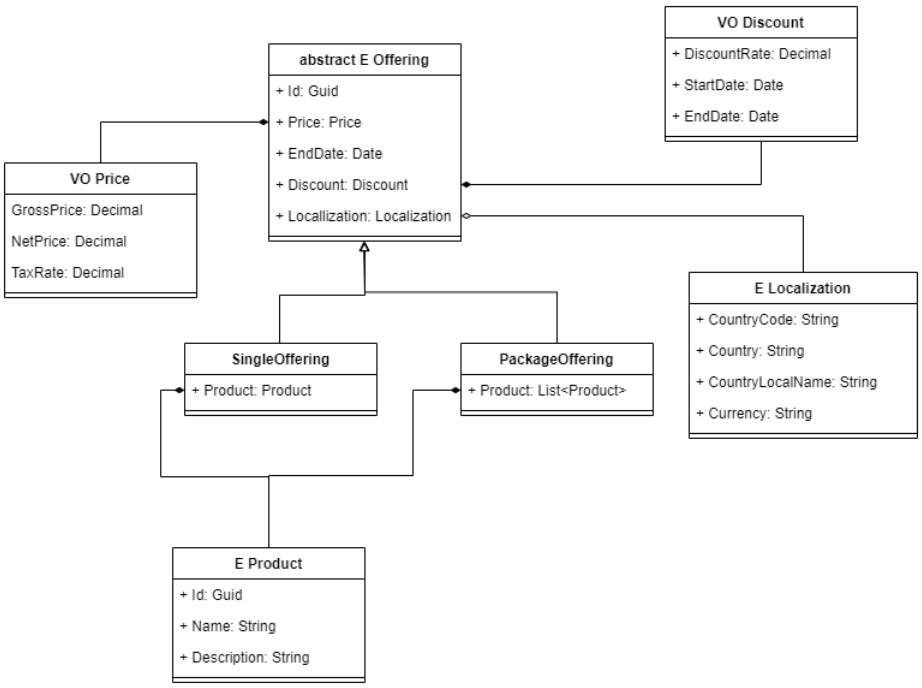


Figure 2 OO Model or Class Diagram of Offer Service step 2

Further the entity will also hold the `Currency`, which are the currency codes defined in ISO 4217 [4]. Since the `Currency` property is located in the `Localization` class, it must be removed from the `Price`. However with the introduction of the `Localization`, a migration becomes essential. This migration involves creating a new Localization table, seeding it with data, and removing the `currency` column from the `price` entity. The currency value is to be preserved, while also establishing a reference from the offer to the corresponding `Localization` entity. This process ensures a seamless integration of the localization feature into the existing system.

5.2 Integration process in general

The following sections will deal with the implementation steps of the two applications. To keep it simple some minor adjustments or bug fix steps will be summed up in the greater context. The integration process for both of the frameworks ASP.NET and Spring Boot share many similarities, although it is important to note they are not identical. To gain a comprehensive overview of the complete project, the next sections deal with the implementation of the complete project and not only the implementation of the ORM frameworks.

These description involves various aspects: initiating the project, developing the domain model and its functionality, setting up a unit test suite, constructing a controller layer, and also the implementation of the ORM frameworks. These steps already mention different layers, the project uses the Clean-Architecture pattern which is a commonly used architecture pattern.

The Clean Architecture pattern is a software design principle that organizes code into distinct layers, where the inner layers encapsulate business logic and core functionali-

[4] https://www.iso.org/iso-4217-currency-codes.htm

ties, while the outer layers handle user interfaces, frameworks, and external dependencies (Robert C. Martin, 2018:p. 211–217). For both core frameworks the clean architecture is a recommended architecture [5] [6]. Due to this facts both projects make use of the Clean-Architecture pattern.

Additionally both use a docker-compose for the hosted database. The `docker-compose.yml` looks like following, some variables are different for each of the projects.

```
version: '3.3'
services:
  db:
    image: mysql:latest
    restart: always
    environment:
      MYSQL_DATABASE: '<db_name>'
      MYSQL_USER: '<db_username>'
      MYSQL_PASSWORD: '<db_password>'
      MYSQL_ROOT_PASSWORD: '<db_root_password>'
    ports:
      - '3306:3306'
    expose:
      - '3306'
    volumes:
      - my-db:/var/lib/mysql
volumes:
  my-db:
```

The exact implementation does not deal with the docker connection in detail and will treat it as any MySQL instance.

5.3 Integration process for the ASP.NET project

In the first steps the initial setup was approached. The initial step involved setting up the basic project. The .NET SDK is shipped with some basic templates for the initial setup. For this project the SDK version 6 was used. To create the Offer service the '.NET Core ASP.NET Core Web Application' template was used. Within the same solution, a .NET Core xUnit project was established using another provided template. At this stage, the application operates as a web server featuring a 'Hello World' endpoint and a test suit.

The next step was the creation of the Domain Model or Domain Layer when considering the Clean Architecture. This was archived by adapting the previously showcased OO model into code. Additionally, some methods are implemented to execute operations, such as adding a discount or calculating gross and net prices. The `Offer` class was one of the classes that were created in this phase.

```
public abstract class Offer
{
    public Guid Id { get; protected set; }
    public string Name { get; protected set; }
```

[5] https://learn.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures

[6] https://www.baeldung.com/spring-boot-clean-architecture

```
    public Price Price { get; protected set; }
    public Discount? Discount { get; protected set; }
    public DateTime StartDate { get; protected set; }
    public DateTime EndDate { get; protected set; }
    protected Offer(Guid id, string name, Price price,
        DateTime startDate, DateTime endDate)
    {
        Id = id;
        Name = name;
        Price = price;
        Discount = null;
        StartDate = startDate;
        EndDate = endDate;
    }
    public void ApplyDiscount(Discount discount)
    {
        if (Discount != null)
        {
            throw new InvalidOperationException("Discount already exists");
        }
        Discount = discount;
        Price = CalculatePriceFromDiscount(discount, Price);
    }
    ...
}
```

This class looks similar to the other classes in terms of OO implementation. The properties are all public gettable but not settable. Altering or assigning a property is only doable for a subclass or through executing a method. Moreover, for each operation, a corresponding test was created in the test project. As the project progressed, minor modifications were made to the domain model.

The next step was in particular relevant to the implementation of the ORM, or more specifically, EF Core in the ASP.NET ecosystem. To start with the implementation the repository pattern was implemented. The repository pattern abstracts the data access layer from the domain layer of an application. Initially two interface repositories were created, `IOfferRepository` and `IProductRepository`, each corresponding to an object that is directly queried from the database. The other objects will be queried thought these.

While these interfaces are not strictly mandatory, they are a usual approach and they serve as a contract for standardized queries. Once this was done, the actual implementation of these interfaces were implemented. Since these implementations needed data access, the EF Core package was installed to leverage its abstraction capabilities. The installation can be performed via a GUI in Visual Studio or Rider or via the CLI command `dotnet add package Microsoft.EntityFrameworkCore -version 7.0.7`. With this set up the actual queries still needed to be written with LINQ queries.

```
public class OfferRepository : IOfferRepository
{
    public DataContext _context { get; set; }

    public OfferRepository(DataContext context)
    {
        _context = context;
    }
    public async Task<List<Offer>> FindAll()
    {
        return await _context.Offers.ToListAsync();
    }
    public async Task<List<Offer>> FindAllAvailable()
    {
        return await _context.Offers.Where(o => o.EndDate > DateTime.Now)
            .ToListAsync();
    }
    public async Task<Offer> FindById(Guid id)
    {
        return await _context.Offers.Include(o => (o as SingleOffer).Product)
            .Include(o => (o as PackageOffer).Products)
            .FirstOrDefaultAsync(po => po.Id == id);
    }
    public async Task<Offer> Add(Offer offer)
    {
        _context.Offers.Add(offer);
        await _context.SaveChangesAsync();
        return await FindById(offer.Id);
    }
    ...
}
```

In this example the `OfferRepository` uses a `DataContext` instance utilizing dependency injection. The DataContext is important, since it is the implementation of the data access layer, provided by EF Core. In this sample code snippet some methods, provided by LINQ can already be observed. The `FindAllAvailable()` method shows some filtering capabilities of LINQ can be noticed. Taking a close look at the `FindById(Guid id)` method, include calls can be seen. These ensure that all references are loaded with the entity, since by default EF Core just lazy loads the entities.

Following the repository implementation the `DataContext` was implemented.

```
public class DataContext : DbContext
{
    public DbSet<Product> Products { get; set; }
    public DbSet<Offer> Offers { get; set; }
    ...
}
```

The `DataContext` inherits the `DbContext` class from EF Core. This class enables the establishment of a database connection and execution of SQL queries. Therefore it is the

actual implementation of the data access. The class also includes two `DbSet` implementations, each corresponding to a table in the later created database. The `DbSet` class also enables CRUD operations on entities, as demonstrated in the code sample of the `OfferRepository` implementation. This sets up the essential groundwork for establishing the repositories and queries, but further configuration is required to complete the functionality of the application.

Before proceeding with the EF Core configuration, a basic controller for offers was introduced. Though this controller wasn't mandatory and could have been added later, it was implemented to manually verify the application functionalities. At that stage of the project, the controller allows for fetching, adding, and removing offers. Once the controller was setup, the only remaining step was the EF Core configuration to launch the application.

The configuration involves mapping the entities using the Fluent API. Simple properties like a `string` or `integer` do not necessitate mapping. However, every reference or more complex data type must undergo configuration as seen in the following example.

```
public class DataContext : DbContext
{
   ...
    protected override void OnModelCreating(ModelBuilder modelBuilder)
        modelBuilder.Entity<Offer>()
            .HasDiscriminator<string>("OfferType")
            .HasValue<SingleOffer>("SingleOffer")
            .HasValue<PackageOffer>("PackageOffer");

        modelBuilder.Entity<Offer>()
            .OwnsOne(o => o.Price);

        modelBuilder.Entity<Offer>()
            .OwnsOne(o => o.Discount);

        modelBuilder.Entity<SingleOffer>()
            .HasOne(so => so.Product)
            .WithMany()
            .HasForeignKey("ProductId")
            .OnDelete(DeleteBehavior.Cascade);

        modelBuilder.Entity<PackageOffer>()
            .HasMany(po => po.Products)
            .WithMany()
            .UsingEntity(j => j.ToTable("PackageOfferProducts"));
    }
}
```

Initially, the `OnModelCreating()` provided by EF Core is overridden. Then all tables that need further mapping, need to be registered with `modelBuilder.Entity<ClassName>()`.

The first explicit configuration involves mapping the `Offer` and its subclasses `SingleOffer` and `PackageOffer`. This implementation uses a Table per Hierarchy (TPH) pattern recommended by the .NET documentation [7]. EF Core also offers implementations for the usage of other patters like Table-per-type (TPT) configuration Table-per-concrete-type configuration (TPC). The following configurations involves embedding the Price and Discount into the `Offer`. This means no explicit table will be created for these entities, instead each of their properties will have a corresponding column in the `Offer` table.

Next, the references between `SingleOffer` and `PackageOffer` to the `Product` were established. The first establishment, `.HasOne(so => so.Product).WithMany()`, results in a foreign key column in the later created Offer table. The `PackageOffer - Product` relation is resolved with a helper table, established throught the `.UsingEntity(j => j.ToTable("PackageOfferProducts"));` call.

With this confiurations done the OO model is mapped to the database. However, additional setup is still necessary. In the application startup file, the repositories and the `DataContext` require registration, which can be accomplished as follows.

```
builder.Services.AddDbContext<DataContext>(options =>
    options.UseMySQL(configuration.GetConnectionString("DefaultConnection")));
builder.Services.AddScoped<IOfferRepository, OfferRepository>();
builder.Services.AddScoped<IProductRepository, ProductRepository>();
builder.Services.AddScoped<ILocalizationRepository, LocalizationRepository>();
```

This code will enable the creation and injection of these services.

Before launching the application and establishing a database connection, migrations were created and executed. To generate a migration, an additional package, `Microsoft .EntityFrameworkCore.Design`, was installed. This package provides the functionality required for creating and executing migrations. With the package installed, an initial migration was created with the command `dotnet ef migrations add InitialMigration`. After the command was executed a new directory Migrations was created. It consists of a file named by the timestamp and name of the migration (`20230615133406_InitialMigration.cs`). This file defines the actual migration steps in C# language. Additionally, the command generates a `DataContextModelSnapshot`, where the current table layout is defined in C#. This file will always update when a new migration is added.

With the migration folder setup its still necessary to execute the migrations. This requires the addition of a connection string to the `appsettings.Development.json` file. This connection string consist of the address to the database and the login credentials for an user. Once the connection string was configured, the migration was executed with the command `dotnet ef database update`. This command implemented the migration on the connected database.

With the successful configuration and executed migrations, the application supported all functionality defined for the first stage and was ready to run. With an http client the database queries were tested manually throught the given endpoints provided by the controller. For more comprehensive testing, integration tests where established. To setup integration tests for EF Core, Microsoft recommends different approaches [8]. Two different

[7] https://learn.microsoft.com/en-gb/ef/core/modeling/inheritance
[8] https://learn.microsoft.com/en-gb/ef/core/testing/choosing-a-testing-strategy

test setups where implemented. One test suite utilizes the Moq NuGet package to mock repositories, while the other replaces the used database with a new test database instance. The mock setup is not further evaluated since it does not test the ORM implementation it just 'mocks' it away. The other approach exchanges the used database instance and therefore provides relevant results for the ORM implementation. To set these tests up initially, a fixture class was created. This class ensures the application can be tested with a changed and active database connection. It's also a good practice to create test data within this fixture.

```
public class TestDatabaseFixture
{
    private const string ConnectionString =
        "server=localhost;database=db;user=offer-user;password=password";
    public TestDatabaseFixture()
    {
        using var context = CreateContext();
        context.Database.EnsureDeleted();
        context.Database.EnsureCreated();
        Cleanup();
    }
    public void Cleanup()
    {
        var context = CreateContext();
        context.RemoveRange(context.Offers);
        context.RemoveRange(context.Products);
        var price = Price.CreateFromGross(10, 19, "EUR");
        var product1 = Product.Create(Guid.NewGuid(), "Product 1", "Description");
        var product2 = Product.Create(Guid.NewGuid(), "Product 2", "Description");
        var offer = SingleOffer.Create("Offer 1", price, DateTime.Now,
            DateTime.Now.AddDays(1), product1);
        context.AddRange(
            product1,
            product2,
            offer
        );
     context.SaveChanges();
    }
    public DataContext CreateContext()
        => new DataContext(new DbContextOptionsBuilder<DataContext>()
                .UseMySQL(ConnectionString).Options);
}
```

To create the tests another class was setup up. It contained the actual tests. At this stage of the project it just contained tests the repository or ORM through the controller method calls. A test suit for testing the repositories directly, was included at a later stage of the application.

```
public class OfferControllerTest : IClassFixture<TestDatabaseFixture>, IDisposable
{
```

```
    public TestDatabaseFixture Fixture { get; }
    public OfferControllerTest(TestDatabaseFixture fixture)
    {
        Fixture = fixture;
    }
    public void Dispose()
    {
        Fixture.Cleanup();
    }
    [Fact]
    public async void ListOffers_Should_Return_All_Offers()
    {
        // Arrange
        var context = Fixture.CreateContext();
        var _offerRepository = new OfferRepository(context);
        var _productRepository = new ProductRepository(context);
        var controller = new OfferController(_offerRepository,
            _productRepository);
        // Act
        var result = await controller.ListOffers();
        // Assert
        var okResult = result as OkObjectResult;
        var value = okResult.Value as List<Offer>;
        Assert.Equal(1, value.Count);
    }
    ...
}
```

With the tests in place and the requirements for the first stage of the development in place, it was proceeded with the second stage.

Subsequently the second project phase was started. The phase consisted of the implementation of the Localization feature. This phase started with the implementation of a `Localization` class without any functionality. Further a repository implementation, to fetch localization entities from the database, were added. To complete the object model, the reference between `Localization` and `Offer` was added as a one-to-one reference and the currency property is removed from the `Price`. With the objects in place, the entities had to be mapped. The new reference is mapped in the `OnModelCreating()` method in the `DataContext` class. Also since there is no property named id in the `Localization`, a identifier key has to be configured in the mapping.

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    ...

    modelBuilder.Entity<Offer>()
        .HasOne(o => o.Localization)
        .WithMany()
```

```
        .HasForeignKey("LocalizationCountryCode")
        .OnDelete(DeleteBehavior.Cascade);

    modelBuilder.Entity<Localization>()
        .HasKey(l => l.CountryCode);
}
```

With this in place a migration was created with a CLI command. But before the migration can be executed some further adjustments need to be setup in the migration. Since the reference between `Localization` and `Offer` need to be setup depending on the current currency, some code has to be added to the generated migration. First some code to seed some localization data is added. Then the reference column `LocalizationCountryCode` was set depending on the still existing `Currency` of the`Offer` and the queried `Localization` with the currency, with a custom SQL.

```
public partial class AddLocalization : Migration
{
    protected override void Up(MigrationBuilder migrationBuilder)
    {

        // ... adding of new column in offer and adding of new table

        migrationBuilder.InsertData(
            table: "Localizations",
            columns: new[] { "CountryCode", "CountryName", "LocalName",
                "Currency" },
            values: new object[,]
            {
                { "DE", "Germany", "Deutschland", "EUR" },
                { "US", "USA", "USA", "USD" },
                { "GB", "Great Britain", "Great Britain", "GBP" },
                { "CH", "Switzerland", "Switzerland", "CHE" },
            });

        migrationBuilder.Sql(@"
            UPDATE Offers SET LocalizationCountryCode = (
                SELECT CountryCode FROM Localizations WHERE Currency =
                    Offers.Price_CurrencyCode
            );
        ");

        // ... creation of the foreign key and the drop of the currency column
    }
}
```

With the tweaked migration, it was executed with the CLI command and the second stage of the project was also completed.

After this stage of the project some minor changes for testing and analyse purposes where added. For example pagination, adding a non existing product in the database through an offer and an extension of the test suit.

5.4 Integration for Spring Boot

To initialy start a new project spring offers the Spring Initializr. The Initializr is an click through UI in the web [9] but is also implemented in the InteliJ IDE. The project was created with the Initializr as Maven project with the java version 17, in the dependency selection Spring Web and Lombok were selected. The Ininializr generated a project source aswell as a test suit.

With this step accomplished, the data models can be created. The created OO model was build pretty similar as in the ASP.NET project. For class implementation the lombok package was used to generate simple code like getters, setters or constructors.

```
@NoArgsConstructor
public abstract class Offer {
    @Getter
    protected UUID id;
    @Getter
    protected String name;
    @Getter
    protected Price price;
    @Getter
    @Nullable
    protected Discount discount;
    @Getter
    protected LocalDateTime startDate;
    @Getter
    protected LocalDateTime endDate;
    protected Offer(UUID id, String name, Price price,
        LocalDateTime startDate, LocalDateTime endDate) {
        this.id = id;
        this.name = name;
        this.price = price;
        this.startDate = startDate;
        this.endDate = endDate;
    }
    public void applyDiscount(Discount discount) {
        if (discount == null) {
            throw new IllegalArgumentException("discount must not be null");
        }
        this.discount = discount;
        this.price = calculatePriceFromDiscount(discount, this.price);
    }
    ...
}
```

[9] https://start.spring.io/

The subsequent step was to map the entities and to configure the database. First the packages necessary for this step were added to the project. The package `spring-boot-starter-data-jpa` and the `mysql-connector-java` were added. The JPA packages is the default implementation of hibernate, the other package is a connector to a MySQL database instance.

For the mapping the different classes and references were marked with the sufficient annotations. In the following example several class examples are shown, it can be observed, that the inheritance is mapped with the `@MappedSuperclass` and the `@Inheritance` annotation. The inheritance strategy used is the TPH pattern (in hibernate called single table), the discriminator can be added in the class implementation with an `@Discriminator annotation` but its not necessary. The other types TPT, and TPC (TPC in spring boot is called joined) are also supported. All other entities or classes that shall be persisted are annotated with an `@Entity`. The classes `Price` and `Discount` were marked as `Embedabble` and when referenced in an entity annotated with an `Embedded`. This ensures that the properties of these classes are owned or embeded whithin that entity. When the actual database table is created these embedded properties are in the entities table. The entity classes also need an identifier, this is a unique property marked with the `@Id`. Then the references were marked with a reference annotation, in this example the `@MantToMany` annotation. There are also annotations for the other types of references `@OneToOne`, `@ManyToOne` and `@OneToMany`.

```
@MappedSuperclass
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@NoArgsConstructor
public abstract class Offer {

    @Id
    @Getter
    protected UUID id;

    @Embedded
    @Getter
    protected Price price;

    @Embedded
    @Getter
    @Nullable
    protected Discount discount;

    ...
}

@Entity
@NoArgsConstructor
public class PackageOffer extends Offer {
    @ManyToMany
    protected List<Product> products;
```

```
    ...
}


@Embeddable
@NoArgsConstructor
public class Price {
...
}
```

When the mapping is setup the database connection credentials where added to the `application.properties` file.

To query the data from the database repositories where created. These repositories were implemented as interfaces that inherit the `JpaRepository`. The `JpaRepository` needs two generic definitions, first the entity that will be queried via the repository and the identifier type of the entity,`Offer` as type and UUID as identifier in the example. The `JpaRepository` already has some definition for simple CRUD operations like `findeAll()`, `save()`, `delete()`, etc. . Other queries can be added as method defnition. If these methods follow a specific naming convention, JPA/Hibernate is able to derive the corresponding SQL queries without having to write them explicitly.

```
public interface IOfferRepository extends JpaRepository<Offer, UUID> {
    List<Offer> findByEndDateGreaterThan(LocalDate date);
    List<Offer> findByEndDateLessThan(LocalDate date);
    List<Offer> findOExpiredOffers();
    List<Offer> findByProductId(UUID id);
}
```

Subsequent to the setup of the repositories a simple controller is added as in the ASP.NET project. There is no more configuration needed to start and test the project manually. It has to be noted that no migrations need to be created as it is neccessary in EF Core. As default on startup the database is dropped and always created newly.

At this stage only some mocked tests where created. With all the requirements of the first step setup the project was ready to run.

Before the second stage of the project could be started some configuration had to be set up. As already mentioned there are no migrations like in EF Core in Hibernate. To be able to created versioned migrations a new package needs to be added. For this project Flyway was added as a maven package. The `flyway-mysql` package is added as dependency and as plugin with configuration.

```
<plugin>
    <groupId>org.flywaydb</groupId>
    <artifactId>flyway-maven-plugin</artifactId>
    <version>9.20.0</version>
    <configuration>
```

```
        <url>Host URL</url>
        <user>userna,e</user>
        <password>password</password>
        <schema>schema-name</schema>
        <location>local path to migration files</location>
        <driver>database-drive</driver>
    </configuration>
</plugin>
```

To create a new migration a new file was created in the local migrations directory, which location is defined in the configuration. Flyway uses the naming convention of `V{version_number}___schema.sql`. In the free and open source variant of Flyway there is no possibility to generate a migration. Due to that fact the a sql script needs to be written manually. For simplicity the sql JPA uses to create the database when starting the server, was copied and used as initial migration. Then the migration was executed with the command `mvn flyway:migrate`.

When the initial migration was setup, the second stage of the project phase was started. The model creation and the adjustment for the localization feature were pretty similar to the ASP.NET project. Further the configurations for the mapping were added to the localization and the new reference was added to the `Offer`. Also a new repository interface for the `Localization` was added. This repository does not need any method definitions, since the CRUD operations implemented by the `JpaRepository` were sufficient.

```
@Entity
public class Localization {
    @Id
    @Getter
    private String countryCode;
    @Getter
    private String countryName;
    @Getter
    private String localName;
    @Getter
    private String currency;
}

@Entity
public class Offer {
    ...

    @Getter
    @ManyToOne
    @JoinColumn(name = "localization_id")
    private Localization localization;

    ...
}
```

```
public interface IProductRepository extends JpaRepository<Product, UUID> {
}
```

The subsequent step after the classes were mapped, was the creation of the migration. For this a new migration file `V2__Add_lcalization.sql` was created. First the new column for the `localizationid` was added to the offer table. Then the localization table was added and afterwards some sample seed data is added. Then the `localizationid` depending on the `Currency` and `Localization` is set. Afterwards the foreign key reference between localizationid and offer is established. And the last step is the deletion of the `currency_code` column.

```
ALTER TABLE offer ADD COLUMN localization_id varchar(255);
CREATE TABLE localization (
                              country_code varchar(255) NOT NULL,
                              country_name varchar(255) NOT NULL,
                              local_name varchar(255) NOT NULL,
                              currency varchar(255) NOT NULL,
                              PRIMARY KEY (country_code)
);
INSERT INTO localization (country_code, country_name, local_name, currency)
    VALUES
    ('DE', 'Germany', 'Deutschland', 'EUR'),
    ('GB', 'United Kingdom', 'United Kingdom', 'GBP'),
    ('US', 'United States', 'United States', 'USD'),
    ('CH', 'Switzerland', 'Switzerland', 'CHE');
UPDATE offer SET localization_id = (
    SELECT country_code FROM localization WHERE currency = offer.currency_code);
ALTER TABLE offer ADD CONSTRAINT fk_offer_localization FOREIGN KEY
    (localization_id) REFERENCES localization(country_code);
ALTER TABLE offer DROP COLUMN currency_code;
```

Afterwards the migration was execution. The Localization feature and also the second stage of the project was accomplished.

The Spring Boot project had some integration tests as well, but setting them up was less effort. To set up a integration test in Spring Boot the Test class just has to be annotated with `@SpringBootTest`. A base class was just used for setting up some general data necessary for all integration tests.

```
public class OfferRepositoryTest extends TestBaseClass {
...
}
```

Just as in the ASP.NET project, after the tests where in place and the requirements set up, some small changes and adjustments were done.

6 Comparison

In the following section, we undertake a comprehensive comparison to show the key differences between EF Core and Hibernate withing the selected topics. This comparative analysis aims to provide valuable insights into their strengths, limitations, and potential use cases.

6.1 Ecosystem

To get a brief overview how the ORMs fit into the ecosystems and how well they are supported, the following subsection analyzes the ecosystems around the ORMs Hibernate and EF Core. Specially the documentation and the community are evaluated.

6.1.1 Documentation

Since .NET is developed by Microsoft, Microsoft offers a very detailed documentation on its website. The .NET documentation is available at GitHub and contribution is welcome (Microsoft, 2023*c*). This documentation is machine-translated, therefore offering accessibility in multiple languages.

The .NET documentation is accurate and comprehensible, serving as a reliable guide for the development of the project.

As the ISO 26514 standard recommends: "Where there is a wide variety of users, with a wide variety of experience, skills, and knowledge, the information for users should be comprehensible (adequate for use) by the least experienced of the expected users" (ISO/IEC, 2022:p.26), the .NET documentation offers documentation for different experienced users. To cover different types of experiences, the .NET documentation can be divided into different kinds of documentation styles:

- **Getting Started**: This section offers some basic introductions to different topics, where first of all new users get information and interactive tutorials. The interactive tutorials often have a code editor built into the website.
- **General Documentation**: This section makes the largest part of the documentation. It dives deeper into specific topics or features, like EF Core. This documentation provides an overview of the general features, dives deeper into specific features and also offers some sample code.
- **API Documentation**: This is a more technical documentation. It describes packages, classes, methods, events, and properties provided by .NET or related frameworks. It also includes descriptions, remarks, constructors, method parameters, return values, and explanations of possible exceptions.

The previous section provided already an breif overview of the .NET documentation's structure. In this section the structure is further evaluated. When visiting the .NET documentation [10], there is a hierarchy of different topics. At the top, there are general subjects such as download guides and interactive tutorials. As navigated further down,

[10] https://learn.microsoft.com/en-us/dotnet/

more specialized topics emerge, ranging from updates in the latest version, concepts, tools, to data access tools. Even further down, the content becomes even more specific, showing language-specific topics and application types that can be developed using .NET. The last section is the API documentation.

Leaving the navigation and analyzing the page structure of an actual documentation page. First of all it can be observed that some pages look differently. This is specially comes from the offer of tutorials and training pages. Some are interactive, others are sample-based, and a others leading to a certification. For the other sub pages the structure becomes more consistent. When navigating into a general documentation, a sidenav on the left[1] lists specific subtopics, while the a sidnav right[2] suggests additional or related topics. The page itself is divided into a header[3], a content overview[4], a concept body[5], subsections[6], illustrations and examples[7]. Through the content are cross-references linking to other relevant topics and API documentation.

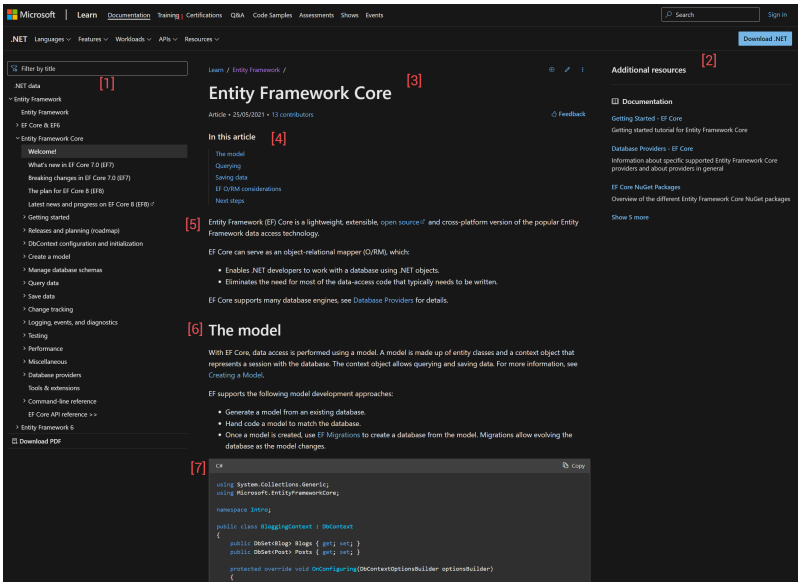


Figure 3 Sample General documentation page

Compared to the .NET documentation, the spring docs [11], structures the root page a bit differently. The root lists all projects within the ecosystem. For instance the Spring Boot project. When accessing the Spring Boot page, it can be minimalist overview consisting of general information, version details, and a learning section cam ne observed. The learning section is similar to the tutorials offered by the .NET documentation. For a more specific documentation the overview links to the reference and api docs (e.g. the Spring Boot references [12]).

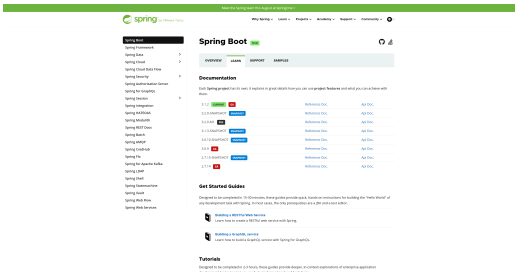
Yet, there already are some contrasts, by taking a look at the actual page content, the contrast becomes greater. Spring Boots sub pages are usually some minimalistic general overview pages. There are no sup pages that have more specific informations like .NET often has. However the content structure across these pages has similarities to .NET. Each page is structured into segments and sections including a header, content overview, main body, subsections with illustrations and examples. A notable difference is that Spring

[11] https://spring.io/projects
[12] https://docs.spring.io/spring-boot/docs/current/reference/html/index.html

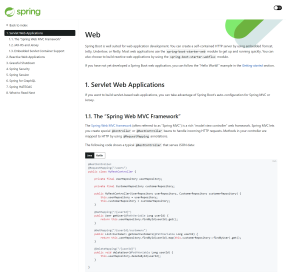
makes less use of cross connections between different topics with links.

Therefore the API documentation content is pretty similar to .NET. They also offer descriptions for the java language specific building blocks. Unlike .NET, however, Spring Boot's documentation is just accessible in English. The GitHub access is also handled differently, Spring Boot has the documentation always in the projects repositories (e.g. the spring boot documentation is in the spring boot repository [13]), while .NET has a repository with most of its documentations. Overall the documentation itself is comprehensible and correct,

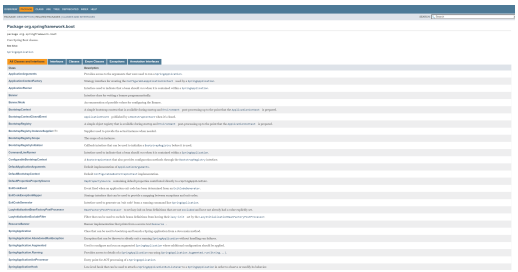
If we take a better look at the Spring documentation structure, it differs from .NET. Unlike .NET, which maintains consistent side layout navigation, Spring Boot adopts three distinct page types. The overview(a), reference docs(b) and API docs(c). The overview and reference docs are navigated using a side navigation (sidenav), while the API docs employ a top navigation (topnav). Interestingly, each page type utilizes a unique font style and layout.



(a) Spring Overview



(b) Spring Boot references



(c) API Documentation

Figure 4 Structure comparison

6.1.2 Community

To compare the community different metrics are compared. Community activity can give a rough estimate of its size. Beginning with analyzing GitHub activity, followed by a comparison of questions and discussions on Stack Overflow. Additionally the Stack Overflow Developer Survey is evaluated within the relevant topics.

Starting with the Github repository comparison. The following GitHub repositories are evaluated:

- (.NET)core [14]

[13] https://github.com/spring-projects/spring-boot
[14] https://github.com/dotnet/core

- aspnetcore[15]
- spring-framework[16]
- spring-boot[17]

In terms of stars, the spring ecosystem leads, the spring-framework has 52.600 stars and spring-boot has 68.600. In contrast, the .NET core repository has 19.200 stars, while aspnetcore has 32.300. The number of forks shows a similar ratio.

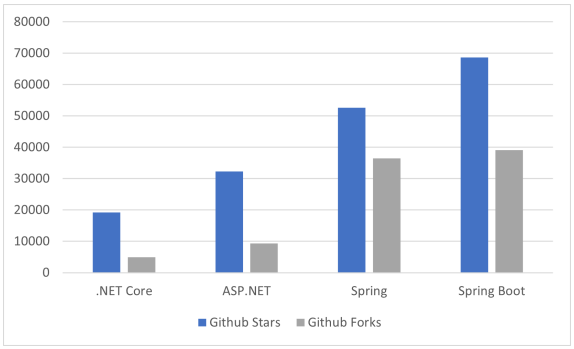
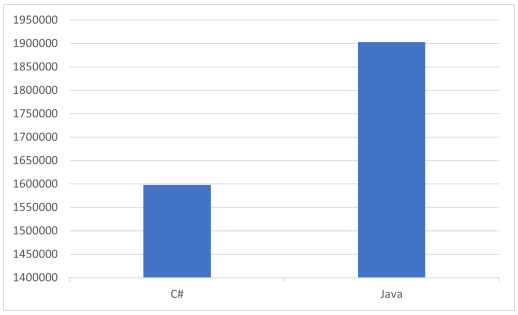
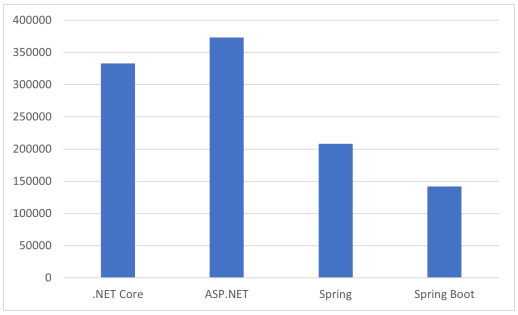


Figure 5 GitHub Stars and Forks

Heading to Stack Overflow, the total number of questions associated with specific tags: .net, asp.net, spring, spring-boot, c#, and java are compared. Regarding the frameworks, the .NET community has 332.990 questions for .net and 373.472 for asp.net. In contrast, spring has 208.037 questions and spring-boot has 141.949. When looking at language specific tags, C# has 1.598.078 questions, while java boasts 1.903.044.



(a) Spring Overview



(b) Spring Boot references

Figure 6 Stack Overflow question count comparison

In the Stack Overflow Developer survey the frameworks and languages where also compared to other languages. The relevant results are extracted.

According to the Stackoverflow 30,5% of participants use Java, while 27,5% use C#. The survey also distinguishes between Professional Developers and developers learning to Code. Among professional developers, Java's percentage remains consistent at 30,5%, but C# usage slightly increases to 29%.

[15] https://github.com/dotnet/aspnetcore
[16] https://github.com/spring-projects/spring-framework
[17] https://github.com/spring-projects/spring-boot

The survey also questioned the usage of frameworks. In this analysis ASP.NET Core as well as ASP.NET are both mentioned. Since ASP.NET Core is the latest version of ASP.NET, ASP.NET is ignored for this analysis. Contrary to GitHub numbers, 16,5% of participants reported that they use ASP.NET Core, and 12% reported using Spring Boot. Among professionals, both frameworks see about a 2% increased usage, with ASP.NET Core at 19% and Spring Boot at 13,5%.

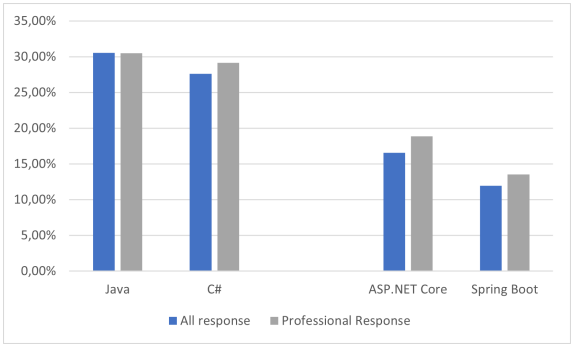
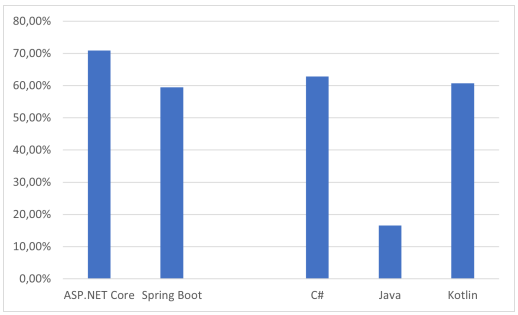


Figure 7 Stackoverflow Developer Survey usage percentages

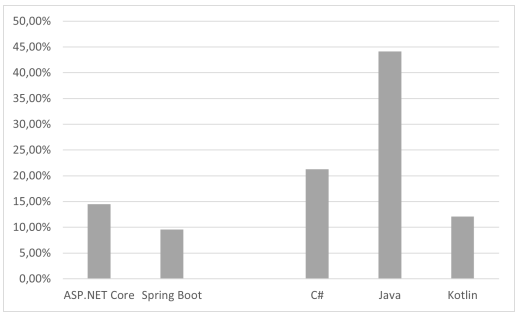
Another interesting analysis is the Admired and Desired analysis, also raised by the Stackoverflow Developer Survey. The Admired and Desired analysis sets, the technology used in the past year and still want to use in the future (admire), in relation with wanting to use technology (desire). Starting here again with the languages. While C# performs with a high value for admire with 69%, Java just has a 16.5% value. Briefly considering Kotlin, which competes with Java, is compatible with the Spring ecosystem and is a more modern language, it still falls behind C# in usage. However, Kotlin's admiration rate stands at 61%, surpassing Java.

For desire, C# leads with 21,5%, followed by Java at 16,5% and Kotlin at 12%.

In the framework category, ASP.NET Core remains popular, still has a high 71% admiration rate. In comparison, Spring Boot has a 59,5% admiration. The Desire values for ASP.NET Core and Spring Boot have scores of 14,5% and 9,5%.



(a) Spring Overview



(b) Spring Boot references

Figure 8 Stackoverflow Developer Survey admire-desire comparisson

6.2 Configuration

For the configuration comparison, different factors are considered. Measuring and analysing the time spent for configuration tasks is difficult because it often overlaped with other tasks, like repository creation. This analysis will focus on the perspective of the ASP.NET project, given the greater configuration steps required compared to Spring Boot and Hibernate. Steps and aspects like required setup packages, database connection processes,

error potential and other essential configurations are considered.

To work with EF Core, first of all the package has to be installed from the NuGet package repository. As described in the implementation chapter, initiating the database requires a migration. The migrations are only executable when the Microsoft.EntityFrameworkCore .Design NuGet package is installed. When trying to migrate it without this package, the error '`Your startup project 'Offering' doesn't reference Microsoft.EntityFrameworkCore .Design. This package is required for the Entity Framework Core Tools to work. Ensure your startup project is correct, install the package, and try again`' is displayed.

For Hibernate on the other hand the first package needed to be installed is the spring-boot-starter-data-jpa. Further the mysql-connector-java package is necessary to connect to a MySQL database instance. When the MySQL connector package is not installed, two errors are displayed in the startup console logs, the first: '`Failed to load driver class com.mysql.cj.jdbc.Driver from HikariConfig class classloader jdk.internal.loader .ClassLoaders$AppClassLoader@1d44bcfa`' and the second error: '`Error creating bean with name 'dataSourceScriptDatabaseInitializer' defined in class path resource ...`' followed by the callstack of internal methods. Compared to the .NET error message, there are some internal class and package names provided by spring in the error log.

To provide the connection parameter in EF Core, its necessary to provide the parameters to the `DbContext`. How exactly this can be done, will be explained in the following paragraphs. For best practise the actual connection string is inserted into the environment variable file (appsettings.Development.json or appsettings.json for .NET) and injected throught that. This looks like following:

```
...
"ConnectionStrings": {
"DefaultConnection": "server=localhost;database=db;user=offer-user;
    password=password"
}
```

If this connection string is not provided, the application will run as usual until the database is accessed. Then error is displayed, with an callstack following.

```
An unhandled exception has occurred while executing the request.
  System.ArgumentNullException: Value cannot be null.
  (Parameter 'connectionString')
  ...
```

If the connection string is provided but the connection to the database cannot be established due to a wrong url or something else, the following error is displayed

```
1. Error: An error occurred using the connection to database 'db' on server
'localhost'
```

```
2. Error: An exception occurred while iterating over the results of a
uery for context type 'Offering.Repositories.DataContext'.
MySql.Data.MySqlClient.MySqlException (0x80004005): Unable to connect to any
of the specified MySQL hosts.
...
```

These two error message are in general a good example for .NET errors. They are easy to read, in the first lines they don't use a lot internal code references and offer a good hint for solving the error.

For the Spring Boot project the steps for the connection establishment are more minimalistic. First of all the different connection parameters need to be added to the environment variable file (application.properties).

```
...
spring.datasource.url=jdbc:mysql://localhost:3307/db-springboot
spring.datasource.username=spring-boot-offer-user
spring.datasource.password=password
```

If this is not provided for the Spring Boot project the error in the console proposes different possibilities to solve the issue:

```
Description: Failed to configure a DataSource: 'url' attribute is
not specified and no embedded datasource could be configured.
Reason: Failed to determine a suitable driver class
Action:
Consider the following:
If you want an embedded database (H2, HSQL or Derby),
            please put it on the classpath.
If you have database settings to be loaded from a
        particular profile you may need to activate it
        (no profiles are currently active).
```

Further if the connection cannot be established, due to a wrong url or the db instance not running the following error with the associated call stack is logged into the console.

```
Communications link failure The last packet sent successfully to the server
was 0 milliseconds ago. The driver has not received any packets from the
server.
...
```

It needs to be said these error message are also easy to read, but the usual error for Spring looks more like the error we previously discovered when the package was not installed. Most errors have a lot of internal code references and no explanation of the error

and are therefore hard to read. However for the Spring Boot project this is all configuration necessary and the application could run with this configuration.

To execute the ASP.NET project, additional setup is necessary because all injected services must be registered as a services in the application's startup file. Thus, repositories are registered as scoped services. To further configure EF Core, the `DataContext` is configured with the `AddDbContext()` method. Right there the connection string is implemented.

```
builder.Services.AddDbContext<DataContext>(options =>
    options.UseMySQL(configuration.GetConnectionString("DefaultConnection")));
builder.Services.AddScoped<IOfferRepository, OfferRepository>();
builder.Services.AddScoped<IProductRepository, ProductRepository>();
```

Not registering a service is a common error. Therefore we take a look at what happens if a service registration is missing. When a service is not registered, an error occurs when the service is tried to be injected. The error is '`An unhandled exception has occurred while executing the request. System.InvalidOperationException: Unable to resolve service for type 'Offering.Repositories.IOfferRepository' while attempting to activate 'Offering.Controllers.OfferController'.`'. This error has no hint to the error itself, but it covers all classes involved in the exception and no internal code references.

For a more general comparison the generated lines of code for both applications are observed. The connection parameters for EF Core can be build into one string and therefore are a bit more minimalistic. However The further configuration necessary is q bit of an overhead.

Topic	**ASP.NET**	**Spring Boot**
Connection Strings	1	3
Further Configuration	4	0
Total	5	3

Table 3 Code lines configuration

6.3 Entity Mapping

During the implementation paragraphs, first insights into the differences of the mapping could already be observed. In this subsection, mapping of classes or entities, simple values, complex values (Value Objects) and references between classes or entities are compared. Further the generated SQL commands for initializing the database are compared.

To start off in general the approach for mapping the entities is completely different, JPA or Hibernate utilize a annotation style within the class implementation of the entity. EF Core on the other side utilizes the Fluent API. This approuch splits the mapping into a new class and therefore the class implementation is free of any mapping. Despite this difference there is one similarity. Primitive data types are mapped automatically, if a class is marked as database entity, it is not necessary to explicitly map these data types.

How to designate a class to be mapped into the database, can be observed in chapter 5. In JPA or Hibernate, this is achieved using the `@Entity` annotation at the top of a class. In contrast, EF Core class get mapped by including them as properties of type `DbSet<T>` in the implementation (`DataContext` in the project) of the `DbContext` class. Each `DbSet<T>` property corresponds to a table in the database where T is the entity type.

As is already clear from the introduction, it is necessary to map complex types or referenced classes manually. There are two different styles for implementing classes or entity on database level. First if a class does not have an identity it is not an explicit entity and needs to be handled differently. It is embedded into the entities table the classes is associated with or a helper table with a foreign key to the entity is generated. The other approach is, that if a class has an identity the reference is resolved with foreign keys and potential helper tables.

For EF Core the classes without identity are mapped thought Owned Entity Types. For this the two methods `OwnsOne` and `OwnsMany` can be used.

In contrast, JPA/Hibernate uses the `@Embeddable`, `@Embedded`,`@ElementCollection` annotations to define and use embedded objects. In chapter 5 it can be observed that a class without identity is annotated with the `@Embeddable` and if its used as a referebce, it is annotated with `@Embedded` or if its a one to many reference with `@ElementCollection`.

Taking a look at the implementation for the inheritance patterns. JPA/Hibernate use the `@MappedSuperclass` and the `@Inheritance` annotation with explicitly naming the inheritance pattern. In EF Core each pattern has to be implemented differently [18], in the case for the case study the TPH pattern was implemented with the call of `.HasDiscriminator(...)`.

In chapter 5 it also could be observed that EF Core does not need an explicit mapping of an identifier. This is done automatically if a property is called `id` or `<EntityName>Id`. This can be overriden at any time with `HasKey` in the `OnModelCreating` method. With a little restriction this also counts for JPA, if the property is called just `Id` is can be mapped automatically. This was not utilized in the case study.

The last comparison is the comparison of the sql snippets execute for database creation. As an example we just take a look at the offer tables. Overall can be said that the created tables have the same tables, columns and references, just named and ordered differently.

[18] https://learn.microsoft.com/en-gb/ef/core/modeling/inheritancetable-per-concrete-type-configuration

EF Core

```
CREATE TABLE ‘Offers‘ (
      ‘Id‘ char(36) NOT NULL,
      ‘Name‘ longtext NOT NULL,
      ‘Price_GrossPrice‘ decimal(18,2)
        NOT NULL,
      ‘Price_NetPrice‘ decimal(18,2)
        NOT NULL,
      ‘Price_TaxRate‘ decimal(18,2)
        NOT NULL,
      ‘Price_CurrencyCode‘ longtext
        NOT NULL,
      ‘Discount_DiscountRate‘ decimal(18,2)
        NULL,
      ‘Discount_StartDate‘ datetime(6) NULL,
      ‘Discount_EndDate‘ datetime(6) NULL,
      ‘StartDate‘ datetime(6) NOT NULL,
      ‘EndDate‘ datetime(6) NOT NULL,
      ‘OfferType‘ longtext NOT NULL,
      ‘ProductId‘ char(36) NULL,
      PRIMARY KEY (‘Id‘),
      CONSTRAINT ‘FK_Offers_Products_ProductId‘
        FOREIGN KEY (‘ProductId‘) REFERENCES
            ‘Products‘ (‘Id‘)
);
```

Hibernate

```
create table offer (
discount_rate float(53) not null,
gross_price float(53) not null,
net_price float(53) not null,
tax_rate float(53) not null,
discount_end_date datetime(6),
discount_start_date datetime(6),
end_date datetime(6),
start_date datetime(6),
id binary(16) not null,
product_id binary(16),
dtype varchar(31) not null,
currency_code varchar(255),
name varchar(255),
primary key (id)) engine=InnoDB;

alter table offer
    add constraint
        FK3cow2cmfxb0nrt43hxm7yu1q3
        foreign key (product_id)
        references product (id);
```

To sum everything up the code generated and time consumed for developing steps including creating the basic domain or object model and mapping this to the database are compared. Just the code generated for the first development step is included. Only the effective code is compared, means that every line just for indentation or empty lines are not taken into account. Also only the delta is used, only the code generated in the step. Overall can be said that total amount of code needed is lower for Spring Boot. But its worth to be noted that the entity mapping itself results in more code. This is ude to the fact that a lot of imports are necessary for JPA/Hibernate mappings. For the time necessary to build the projects, Spring Boot also has a slight lead by 10 minutes.

Step	**ASP.NET Effective Code**	**Spring Boot Effective Code**
Build Domain	69	109
Map entities	19	29
Total	88	138

Table 4 Code comparison for entity mapping

6.4 Query Building

This section aims to compare the querying capabilities of JPA/Hibernate and EF Core, focusing on their query syntax, expressive power and functionality.

If we start with Hibernate, Hibernate offers two different approaches, the first based on naming convention Derived Query Methods features as already seen in chapter 5, but

Step	ASP.NET Time	Spring Boot Time
Build Domain	47mins	43mins
Map entities	30mins	24mins
Total	1h 17mins	1h 7mins

Table 5 Time comparison for entity mapping

further also the possibility to write queries with Hibernate Query Language (HQL). As it can be observed in the sample code from the case study in chapter 5, most simple queries can be build with the Spring Data Query Methods. Derived Query Methods have multiple keywords that can be used, some common of these are:

- `find`: Used to retrieve data from the database. It can be followed by a `By`, and then the field name or property on which the filter is applied on
- `count`: Used for counting the number of records that match the specified criteria. It also can be followed by By and then the field name or property on which you to filter
- `delete`: Used to delete records from the database. It can also be followed by By and then the field name or property on which to filter
- `exists`: Used to check if records matching the specified criteria exist
- `findAll`: Used to retrieve all records from the database
- `OrderBy`: Used for ordering the results based on a specific field.
- `Between`: Used for specifying a range between two values for filtering
- `LessThan`, `LessThanEqual`, `GreaterThan`, `GreaterThanEqual`: Used for specifying comparison operators in filtering
- `IsNull`, `IsNotNull`: Used for checking null or non-null values in filtering
- `and`: used for chaining multiple commands with a logical and
- `or`: also used for chaining commands, but with a logical or

These are just some example commands, there are a lot more. The result also depends on parameters and specially the return type. The parameters are import mainly when filtering for a specific property, for example when using `List<Offer>findByEndDateLessThan (LocalDateTime date)`, the result is filtered by the given date parameter. Further pagination and sorting is also possible to use with only the parameters. When adding a `PageRequest` parameter the query is paginated, `List<Offer>findAllBy(PageRequest pageable);`.

Its also possible to query with HQL. HQL is similar to SQL, but it uses the names of the Java entity classes and their properties instead of database table and column names, it also usually has a strong autocomplete support. When adding the HQL with the `@Query`

annotation the HQL command can be written. For example `@Query("SELECT o FROM SingleOffer o WHERE o.product.id = :productId") List<Offer>findSingleOffersByProductId (UUID productId);`.

Further it needs to be noted that some regularly used queries like, finding by Id, finding all, updating by id deleting by id, creating a new entity etc., are already build in and don't need to be created manually.

Regarding expressive power, Hibernate's HQL and Spring Derived Query Methods features are proving to be flexible, allowing developers to express querying logic effortlessly. However, the queries are not reusable and complex queries might get hard to read when written with Derived Query Methods.

EF Core on the other uses LINQ queries. LINQ is an integrated C# language feature. LINQ filter commands/methods can be chained. They usually result in a a `IQueriable`. This `IQueriable` can be resolved into an actual result with terminal or executing methods, these will query and retrieve the data from the database. Further LINQ also provides methods for projection, casting, mapping and joining. Some sample methods are:

- `Where`: Filters a sequence of elements based on a C# condition
- `Take`: Returns a specified number of elements from the start of a list
- `Skip`: Skips a specified number of elements from the start of a list and returns the remaining elements
- `First` / `FirstOrDefault`: Returns the first element that matches a specified condition, or a default value if no match is found
- `Last` / `LastOrDefault`: Returns the last element that matches a specified condition, or a default value if no match is found
- `ToList`: Converts the query result to a C# List
- `Count`: Returns the number of elements in the query result
- `Sum`: Computes the sum of the values in the query result
- `FirstOrDefault`: Retrieves the first element from the query result or a default value if the result is empty
- `Select`: Transforms each element of a sequence into a new form
- `Include`: eager loads the given property

As in the Hibernate example these are just some examples and there are a lot more methods. For common tasks, EF Core does not provide any default implementation as Hibernate does.

Regarding readability EF Core queries also demonstrated expressiveness and conciseness. The necessary implementation gives the advantage, that methods or queries are reusable.

Overall LINQ has a good readability, but its completely different to SQL and therefore specially complex queries might be hard to be translated from SQL to LINQ. At some point it might be necessary to use SQL with `.FromSql()` and the benefit of the LINQ queries is lost.

Looking at potential errors. Due to the fact that the LINQ queries as well as HQL are hard type based and both have a great auto completion support in common IDE's, an error due to a wrong filter or query is very unusual. Syntax errors in the Derived Query Methods may be more likely, but an error with a reference to the method is thrown at build time.

To sum everything up with the simplicity of code generation of the naming conventions. This does also reflect when looking at generated code and time intensity for developing the case study.

Step	**ASP.NET Effective Code**	**Spring Boot Effective Code**
Build Repository interfaces/contracts	19	17
Implement Interfaces	59	-/-
Total	78	17

Table 6 Code comparison for query building

Step	**ASP.NET Time**	**Spring Boot Time**
Build Repository interfaces/contracts	6mins	11mins
Implement Interfaces	22mins	-/-
Total	28mins	11mins

Table 7 Time comparison for query building

6.5 Migrations

As previosly explained, Hibernate does not bring a migration system. The only thing close to that is, the database initialization behavior. There are four different behaviors none, validate, update and create-drop, it can be set in the application properties with the `spring.jpa.hibernate.ddl-auto` property.Validate checks if the present database has an matching schema with the defined mapping in the classes, none does nothing, update automatically makes the necessary changes to the database to match the entity mapping and create-drop creates a new database on startup and deletes it on shutdown.

To get migration functionalities a third party library is necessary. As already mentioned previously Flyway was chosen for the case study. As already explained the open source version of Flyway brings functionality for baseline, clean, info, migrate, repair and validating. Every migration needs to be created manually in a file with special file naming convention. The convention contains the version number and the name of the migration looking like: `V2__Add_lcalization.sql`. Within the file the migration has to be written manually in SQL.

EF Core on the other hand already has a build in feature for migrations. To be able to run these the `Microsoft.EntityFrameworkCore.Design` package has to be added. EF Cores migrations have similar base functionality compared to Flyway. Migrations can be executed, baselined and info's can be shown. Further the migrations can be created automatically. But its also possible to create an empty migration and to write them manually. The migrations are written in a code first approach meaning they are written in C# and not SQL. For most database migrations, no SQL needs to be written, but its also possible to execute SQL commands. The migrations are created with the cli, the created migration files consists of a timestamp and the given name, looks like: `20230623081557_AddLocalization.cs`.

Further EF Core provides the functionality to revert a migration. Each generated migration contains an `Up()` method, which executes the actuall migration and a `Down()` method to undo it. When a migration is automatically generated, the `Down()` method is also created. However, for custom or modified migrations, the `Down()` method must be manually implemented to ensure the rollback functionality. To revert a migration the usual command for running a migration is called, but with a previous version. For example `dotnet ef database update AddLocalization` would revert every migration after `AddLocalization`.

For a side by side comparison we compare the development steps for creating the second stage of the project, the localization feature. For Spring Boot the time for setting up Flyway is also taken into account. In the ASP.NET project most of the new code contains of the migrations. The time necessary to set up the migration tool (Flyway) took a lot of time and therefore the implementation of the localization into Spring Boot took more time. The lines of code therefore are less compared to ASP.NET. The implementation of the migration with C# instead of SQL brings some overhead.

Step	**ASP.NET Effective Code**	**Spring Boot Effective Code**
Flyway setup	-/-	13
Feature development and creation of migrations	279	27
Total	279	40

Table 8 Code comparison for localization feature and migration

Step	**ASP.NET Effective Code**	**Spring Boot Effective Code**
Flyway setup	-/-	2h 23min
Feature development and creation of migrations	2h	1h 38min
Total	2h	4h 1min

Table 9 Time comparison for localization feature and migration

7 Discussion

In the previous chapters provided the results gathered from the comparison and the case study. In this chapter, these results are discussed and put into the context. In addition, some suggestions for improvement and ideas for further studies or analyses are given.

7.1 Ecosystem

7.1.1 Documentation

When comparing ecosystems, especially documentation, .NET's documentation clearly stands out in certain areas. For the so important information quality by (ISO/IEC, 2022:p.26-27), both have a good quality. But .NET stands out due to its multi language availability. Even though English is most common in the software development sector, having multi-language support makes it easier for non native speakers and also people with less experience and less technical knowledge.

Taking a look at the structure .NET documentation is consistent and structured, making it straightforward to use and can also be accessed throught the same web page. On the other hand Spring handles this differently, the navigation and structure throughout a single page is the same. But when using the Spring documentation, different page styles are used and therefore its more challenging to navigate the documentation. These variations can be harder to navigate and understand compared to the well structured documentation provided by Microsoft.

7.1.2 Community

Evaluating community sizes and preferences for languages and frameworks is bit difficult since the variance specially due to the difference findings for user base sizes. By assessing GitHub stars and forks, the Spring ecosystem is dominant. This trend is mirrored in the higher number of Java questions on GitHub. However, there's a shift when considering the .NET ecosystem questions to spring, there are way more questions for .NET than for Spring. The Stack Overflow Developer Survey strengthens this, indicating greater C# and ASP.NET usage compared to Java and Spring Boot. While metrics diverge, both ecosystems clearly have large communities. This includes numerous threads, resolved questions and discussions helping at problem solving.

Reviewing the admiration and desire metrics, the .NET ecosystem looks way better, with Java lagging behind C#. This could be due to C# being perceived as a more modern language than Java.

As a furhter note it should be taken into account that Spring community should have created more third party documentation, like blogs, videos and other guides, due to its long standing presence compared to .NET.

7.2 Code specific

The simplicity Spring and Hibernate aim to provide, can specially be observed in their minimalist code necessary. Even for basic configurations, .NET/EF Core projects require

more lines, largely due to repository registrations and more extensive configuration. This difference is likely to grow for even larger projects. While .NET/EF Core demands more configuration, its error messages are typically more clear and straightforward. On the other hand, Spring Boot error messages can sometimes be less intuitive, often starting with internal package files, making them less user-friendly for understanding the error cause.

The most significant differences are in the Entity Mapping and Query Building. Particularly in the the simple creation of domain models, the features of the C# language stand out. When writing basic classes in C#, it requires less code compared to Java. Taking a specific look at the Spring project code it can be observed that a lot code for imports is necessary. Each annotation needs its import. On the other side, in EF Core, imports (or 'using' in C#) are already less, and also due to the fact that the complete mapping is done in a single file all imports for mappings are only necessary once. Modern IDE's often generate these imports automatically, eliminating the need for manual input. It should be noted that this auto generated code still requires maintenance. If there's a change or removal in mapping annotation, its corresponding import must be manually updated or removed. Therfore for JPA/Hibernate project the chance for dead code is higher.

Looking at the ISO25010 definitions for functionality in terms of the case study, both offered every feature necessary to fulfill the required mappings, with a look at the documentation both still have a lot more to offer, also for more complex projects. Looking at the usability and specially the learnability two divided way for EF Core is a bit more complicated compared to the everything at one place for JPA/Hibernate. If a lot of entities need to be mapped the `OnModelCreating` method can get pretty long and messy. This can be resolved by dividing the entity mapping into new files, but with this approach for every entity another file is created. Which leads into more code to be maintained.

Looking at the auto generated table or SQL by JPA/Hibernate, they are a bit of a mess. While EF Core's generated table colums are ordered like the properties in the class for JPA/Hibernate this looks like a random order. The generated constraints for the the tables for EF Core are named like the associated tables and columns, while JPA just generates some random name. Therfore using the table with an GUI, EF Core generated tables are easier to use.

Regarding the query building Hibernate is more minimalistic compared to EF Core. While its necessary to write actual implementations of EF Core repositories Hibernate only comes with simple interface and Derived Query Methods. Writing the interfaces for EF Core is just good practise, but even without these 19 lines interface code, the 59 line necessary for EF Core is way more than the 17 for Hibernate. The also predefined queries for finding all, deleting, updating, creating etc. come in handy for Hibernate. These offered queries are some base functionality that is necessary for pretty much every application. In an EF Core project these have to be implemented manually. This simplicity is also reflected

in the time necessary for setting up the repositories.

These facts look like Hibernate has the clear lead now, but what this additional expanse for EF Core brings is more flexiblity. Its easier to share code and its easier to adjust single queries. The setting up of eager or lazy loading can be done for every query specifically, while the setting for that in Hibernate needs to be done in the object reference setup and therefore is effective for any query. This fact could be beneficial specially for larger projects, where more queries and more granular control is necessary

EF Core migrations offer seamless integration with .NET applications, making the development experience for .NET developers easier. The code first approach of EF Core migrations also gives a more natural collaboration between application code, database schema and migration code, which helps for maintainability. For Hibernate the seamless integration does not quite apply even tho, Flyway works well its not implemented with Hibernate. Therefore an additional setup process needs to be done. However when setup Flyway integrates well with Hibernate.

Looking at the migrations generated by EF Core its compared to Flyways migrations way more code. This comes due to the fact that EF Core utilizes the code first approach and all migrations are primarily C# code. Flyway's simple SQL files has some simplicity but they have to be written manually and therefore compared to the generated migrations of EF Core, offers great vulnerability to errors.

Further it should be noted, that the test setup for integration tests including a database, is simpler for Spring and Hibernate. While for Spring a test simply has to be annotated `@SpringBootTest` in .NET a fixture class has to be written, which configures the database connection for the tests.

In general in pretty much all code relevant topics Spring Boot and Hibernate offer way more simplicity and minimalism compared to .NET and EF Core. Therefore the configuration over head for .NET and EF Core lead to more flexibility. Its hard to say which of these big benefits way more, in the end everyone should factor these benefits for the planned project and choose the right framework for the specific use case.

7.3 Improvements and limitations

The implementation and analysis covered multiple important topics and features for an ORM implementation. With the ecosystem, entity mapping , querying and migrations, four crucial topics specially for a new project are compared. But there are several improvements and topics a further analysis could continue.

7.3.1 Transactions

Transactions are a fundamental concept in DBMS that ensure data consistency and integrity. A transaction is a sequence of one or more database operations that are executed as a single unit of work. These operations can be reading or writing data into the database. The main purpose of using transactions is to guarantee that all operations within the

transaction are either fully commited or entirely rolled back if an error occurs. An ORM should handle transactions effectively, allowing developers to commit or rollback changes as needed.

7.3.2 Cache

Caching is the practice of storing frequently accessed data in a fast access storage, such as memory, to reduce the need to fetch the same data from the underlying database repeatedly. Setting up a good caching can lead to improved performance ,reduced database load and minimized network traffic. These benefits can lead to a better scalability for an application. Therefore developers should be able to configure the ORM's caching mechanisms.

7.3.3 Project scale

The project or case study this analysis occurred of was just a small scaled service of a broader microservice system. Some requirements and implementations may change if the project scale is getting larger. The differences of generated code may outweigh more, or the flexibility of EF Core may come to shine if more or more complex queries or architectures are necessary.

7.3.4 Different Solutions

For some topics there are different solutions that could also be analyzed. For example Hibernate mapping could also be done with a xml file. EF Core for example also has a support for a annotation style mapping. The selected topics and approaches for this analyses where on best practises and documentation recommendations. But a further analyse could also include the other possible approuches.

Time metrics

It definitely has to be said that before the implementation more experience for the ASP.NET and C# lay ahead. Specially this is a a factor that should kept in mind when looking at the time necessary for development. Some problems for the ASP.NET project where known and fixing them was therefore easier, compared to errors in the Spring project. Another factor that is time relevant is, that both projects where developed asynchronously, the ASP.NET project was developed before the Spring project. Therefore some architecture adjustments and decisions where made while developing the ASP.NET project, that could ve lead to a slight longer development there. Both factors should be kept in mind before comparing the times solely, they should always be interpreted together with the code and explanations aside.

8 Conclusion

8.1 Summary of the main findings and conclusions

In this analysis between .NET's EF Core and Spring Boot's Hibernate, primarily six aspects are evaluated and compared: documentation, community, configuration, entity mapping, querying data and migrations.

Documentation: In terms of documentation, both EF Core and Hibernate have comprehensive resources. EF Core benefits from the fact that its developed by Microsoft, resulting in a well structured documentation and a great accessibility with different examples and a wide range of different tutorials.

On the other hand, Spring Boot's documentation has decent scope, covering various aspects of its usage. It provides detailed explanations of its features, though it might be challenging to navigate, when using it for the first time. However, its long standing presence means a lot of third-party tutorials, blogs, and guides have supplemented the official documentation, offering a lot of materials for learning and troubleshooting.

Community: .NET and Spring both have a large user bases. Both frameworks have an active community, which is critical for knowledge sharing and resolving issues. Nevertheless, even though Spring Boot is a bit older, .NET has caught up and it looks like that compared to Java, C# is way more popular as a language.

Configuration Setup: .NET's and EF Cores configuration can be a bit more workload due to the fact that there is no automatic configurations, like in Spring Boot. When it comes to errors specially in the configuration, .NET's error and exception messages are way clearer and easier to read, compared to Spring Boot's and therefore easier to be resolved.

Entity Mapping: EF Core provides the Fluent API technique for configuring and mapping entities. The `DbContext` class is at the core of EF Core, acting as a bridge between the domain or entity classes and the database. Hibernate or JPA on the other hand use the annotation based configurations, which maps the entity inside the class itself. Both are two different styles of mapping, both are great to use, however due to necessaty of atleast one more file, .NET comes with an overhead.

Querying Data: EF Core uses LINQ for querying data, which provides strong typing and is a C# language feature. However, complex queries might be challenging to implement due to the complexity in translating SQL to LINQ expressions.

Hibernate's Derived Query Methods or HQL offer two different approaches for querying data. Specially the Derived Query Methods are usefull and offer a minimalistic approach for most simple queries. While HQL supports object-oriented features, it can help for complex queries.

Migrations: EF Core's migration feature enables transforming domain/class models into database schemas, supporting both automatic and manual (code-based) migrations. The process is generally straightforward, with CLI tools to handle migrations. Thanks to the code first approach, allowing developers to define database schema through C#, which then EF Core translates into SQL. However this code first approach leads in some more code that has to be written compared to a SQL script doing the same.

Hibernate doesn't natively support migrations. Third party tools like Flyway are commonly used in combination with Hibernate to manage database version control and migrations. Flyway has a good integration with Hibernate. but specially the open source version has less features compared to EF Core's migration system.

In summary, both EF Core and Hibernate are powerful ORM tools, each with its own strengths. Both integrate well into their respective framework. Both bring different strength and weaknesses, considering these can influence the choice of the core framework.

8.2 Recommendations

The choice between EF Core and Hibernate or mre likely .NET and Spring Boot largely depend on the context of the development project and the skill set of the team or developers. Specially the developers or teams experience should play a large role in the decision. If the team is already working in a .NET environment, using C#, EF Core would be the natural choice. Conversely, if the project team is based on Java, Spring Boot would likely be the better option.

For simpler projects it may be attractive to use Spring Boot and Hibernate due to the ease of setup and use.

An issue for EF Core could be the potential of complex queries. While LINQ provides a powerful way to query data its completely different to SQL and therefore a query build in SQL is not always easy to translate to SQL.

Another issue for EF Core could occur with the more workload and code generated specially for larger projects. Every line of code needs some kind of maintainability. The maintainability should also kept in mind for the many necessary imports in Spring Boot.

For Hibernate the lack of migration feature is a big negative point. Migrations are a common approach in modern development and therefore regularly used. Flyway and probably other third party tools may have a good integration but they are another third party tool that can be some kind of a risk.

The documentation should also be considered. If the developer prefers easy to navigate, direct, and beginner-friendly documentation, .NET may be a better fit. However, if the team is comfortable diving into more complex material and using a wide range of third-party resources, Spring Boot could also be choice.

9 List of Abbreviations

ORM	Object Relational Mapping
RDBMS	Relational Database Management System
OO	Object Orientated
OOP	Object Orientated Programming
DDD	Domain Driven Design
CRUD	Create, Read, Update, Delete
HQL	Hibernate Query Language

List of Tables

List of Figures

References

Chen, C., Sun, Y. and Tan, J. (2020), 'Research on application model of marine geological sampling model based on object-oriented relational mapping', *IOP Conf. Ser.: Earth Environ. Sci. 558 032021* .

db-engines (2023), 'DB-Engines Ranking', `https://db-engines.com/en/ranking`. Accessed July 15, 2023.

E.F. Codd (1970), 'A relational model of data for large shared data banks'.

Frederik Wulf (2023), 'Vii.-future-ideas'. Accessed August 08, 2023.
URL: *https://github.com/fredyyy998/ecommerce/wiki/VII.-Future-Ideas*

IBM (2023), 'Java Persistence API (JPA)', `https://www.ibm.com/docs/en/was-liberty/base?topic=overview-java-persistence-api-jpa`. Accessed July 15, 2023.

ISO/IEC (2011), 'Systems and software engineering – systems and software quality requirements and evaluation (square) – system and software quality models'. ISO/IEC 25010:2011.
URL: *https://www.iso.org/standard/35733.html*

ISO/IEC (2022), 'Systems and software engineering — requirements for designers and developers of user documentation'. ISO/IEC 26514:2022.
URL: *https://www.iso.org/standard/77451.html*

JBoss (2023*a*), 'Hibernate ORM', `https://hibernate.org/orm/`. Accessed July 15, 2023.

JBoss (2023*b*), 'What is Object/Relational Mapping?', `https://hibernate.org/orm/what-is-an-orm/`. Accessed July 13, 2023.

Mandapuram, M. and Hosen, M. F. (2018), 'The Object-Oriented Database Management System versus the Relational Database Management System: A Comparison', **7**(2), 89—-96.

Marks, R. M. and Sterritt, R. (2013), 'A metadata driven approach to performing complex heterogeneous database schema migrations', *Innovations in Systems and Software Engineering* .

Microsoft (2022*a*), 'Entity Framework Core', `https://learn.microsoft.com/de-de/ef/core/`. Accessed July 15, 2023.

Microsoft (2022*b*), 'Overview of ASP.NET Core', `https://learn.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-7.0`. Accessed July 15, 2023.

Microsoft (2023*a*), 'ADO.NET Overview', `https://learn.microsoft.com/en-gb/dotnet/framework/data/adonet/ado-net-overview`. Accessed July 15, 2023.

Microsoft (2023*b*), 'Inheritance', `https://learn.microsoft.com/en-us/ef/core/modeling/inheritance#table-per-concrete-type-configuration`. Accessed July 13, 2023.

Microsoft (2023*c*), '.net docs'. Accessed July 29, 2023.
URL: *https://github.com/dotnet/docs*

Microsoft (2023*d*), 'What is .NET? Introduction and overview', `https://learn.microsoft.com/en-us/dotnet/core/introduction`. Accessed July 14, 2023.

mvnrepository (n.d.), `https://mvnrepository.com/open-source/orm`. Accessed July 15, 2023.

nuget.org (2023), 'Microsoft.EntityFrameworkCore', `https://www.nuget.org/packages/Microsoft.EntityFrameworkCore`. Accessed July 15, 2023.

Oracle (2023), 'Java JDBC API', `https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/`. Accessed July 15, 2023.

Redgate (2023), 'Documentation', `https://flywaydb.org/documentation/`. Accessed July 18, 2023.

Robert C. Martin (2018), *Clean Architecture: Das Praxis-Handbuch für professionelles SoftwaredesignRegeln und Paradigmen für effiziente Softwarestrukturierung*, Frechen: mitp-Verlag.

Stackoverflow (2023), '2023 developer survey'. Accessed July 29, 2023.
URL: *https://survey.stackoverflow.co/2023/*

Vial, G. (2018), 'Lessons in Persisting Object Data Using Object-Relational Mapping', **36**(6), 43 – 52.

VMware (2023*a*), 'Spring boot'. Accessed August 08, 2023.
URL: *https://spring.io/projects/spring-boot*

VMware (2023*b*), 'Spring Framework Overview', `https://docs.spring.io/spring-framework/reference/overview.html`. Accessed July 14, 2023.

VMware (2023*c*), 'What spring can do'. Accessed August 07, 2023.
URL: *https://spring.io/*