Applied Research

19/05/2022 Eindhoven
Version: 3.0

S3 Group 1

Members:
Aleksej Borisov: 2776286
Oleksandr Gurianov: 4178092
Mohammad Nazibul Khan: 4263308
Lars Kluijtmans: 4220269
Noelia Rodriguez Morales: 3635988
Esther Wolfs: 3329984

Tutor:
Nicole Zuurbier

Version history

Version	Date	Author(s)	Changes	State
0.1	19/05/2022	Esther Wolfs	Add questions and make structure	Finished
1.1	19/05/2022	Esther Wolfs	Answer question 2.2, 2.3	Started
1.2	20/05/2022	Esther Wolfs	Answer question 2.3	Finished
1.3	23/05/2022	Esther Wolfs	Answer question 2.4	Finished
2.0	28/05/2022	Aleksej Borisov	Answer question 2.1	Finished
3.0	23/06/2022	Esther Wolfs	Add conclusion	Finished

1. Introduction

In this document you will find the applied research we have done during this sprint. The topic we chose is database, because up until now we have only ever worked with relational databases. For this project we were asked to use MongoDB, so we wanted to do some research on it.

2. What is the best database for this project?

2.1 What is the difference between a relational database and a non-relational database

Research Method: Literature Study

Data is a collection of structured information stored in a computer system. The vast majority of all databases fall into two main categories: relational and non-relational. The ways these types are organized are different.

Name "relational" speaks for itself, these kinds of databases store information pieces with pre-defined relationships between them. These pieces are organized as a set of tables with columns and rows. Generally, each table stores information about objects of a single type. The column in the table can be seen as a container which holds attribute values of a single type, whereas the row represents a collection of attributes for a single object (entity). Usually, each row has a unique identifier which is called the primary key. Rows from another table can refer to it by using foreign key (unique identifier of an entity in another table).

Advantages:
1. Data is well structured into categories and is consistent, which makes navigation easier.
2. It is possible to set relations between different tables.

Disadvantages:
1. Limitations. While setting the database it is needed to specify such information as data volume for each field, number of columns, field types, etc.
2. The complexity of the database grows when the amount of data increases.
3. Performance. The response to a query will be slower if there are more tables/rows.

On the other hand, non-relational databases are completely opposite, because they store data in a non-tabular form (data that is not formatted in a table).
Non-relational databases can be based on data structures like documents with unique key id. Each document field value can be a scalar item, like number and text or compound element like list and parent/child collection. The data in the fields of a document can be encoded in various ways: XML, YAML, JSON, BSON, or even stored as plain text. Document can store all the data of a single object (entity), not spreading all the information across different tables.

Advantages:

1. Data model is flexible. It is possible to store and combine different types of data.
2. It is possible to dynamically change your database structure without affecting already existing data.
3. it is possible to expand your database at relatively low cost.
4. High performance. Database is capable of processing a huge amount of information pieces per given amount of time.

Disadvantages:
1. Inconsistency. It is insecure in terms of interacting with data due to high flexibility.

2.2 What information needs to be stored

Research Method: Problem Analysis

Before it is possible to decide how to store information, it should be clear what kind of data is necessary. There are several different parts of the application, all of which need data. To answer this question, the application will be divided in two main parts: the user and the packages.

The application has two different types of users: the creator and the customer. For all of these users some basic information needs to be stored, like their first name and last name. The creators can upload packages to the marketplace, customers can then download these packages. The upload and download information needs to be stored as well. Customers can also leave reviews on the packages they have downloaded, creators can respond to these reviews.

The second main part of the application consists of the packages. For the package itself the obvious things that need to be stored are the name of the package, a description telling the user what the package is about, an icon to display and a list of screenshots.
However, there is more information that is related to the packages. The package needs information on which creator uploaded it. Whenever a creator makes a change to the package, a new version is stored. The creator of the app can also choose to remove the package, but because other packages can depend on that package and because customers can have it downloaded on their device, the package shouldn't be removed entirely. This way there should be a status and the creator should be able to discontinue a package, which would keep all of its information, but prevents new customers from downloading it.
To keep the quality of the packages high, creators need to know certain analytical data like the amount of downloads and ratings and reviews for the packages.

2.3 How to store this information

Research Method: Literature Study

The users and packages share a lot of information, so it is important that everything is stored in the best way possible. A good database design is critical, especially because it is very different from a relational database.

In a relational database you split the data into different tables, so you don't duplicate your data. You can then join the tables together by using foreign keys. When designing a MongoDB schema there is no formal process, there are no algorithms and no rules. It is important to design a schema that specifically works well for the Ivanti App Market application. Instead of splitting the data into separate collections or documents, you can embed data into arrays and objects within the object you created (Karlsson, 2020).

All the user information, like the account information, the roles and the name of the user, can be stored in one "user" document. This way all the data of that specific user can be retrieved with a single query.

A user has a list of packages they have either uploaded or downloaded, and every package also has a creator that has made it. This is where the choice between embedding or referencing has to be made. An embedded document is a type of document that contains a document inside another document (Guide, 2021). When referencing a document, the id of the document that you want to reference is stored in the other document, instead of the entire document. When the application needs the information of the referenced document it runs a second query to return the related data (MongoDB, 2021).

There are certain advantages and disadvantages of both referencing and embedding. The advantages of embedding are that all relevant information is retrievable with a single query, it is not necessary to implement joins in either application code or by using $lookup and updating related information can be done as a single atomic operation. Having large documents comes with some disadvantages: it means having more overhead if most fields are not relevant, limiting the size of the documents improves query performance. The documents for MongoDB also have a size limit of 16MB, so storing a lot of data is not ideal (Karlsson, 2020).

When using referencing, the data is split up between multiple schemas. Some advantages are that the size of the documents is a lot smaller, infrequently accessed information is not needed in every query and it reduces the amount of data duplication, however data duplication should not be avoided if it results in a better schema. The limitation is that two a minimum of two queries are always needed to get all the information (Karlsson, 2020).

A relational database has many different types of relationships, in MongoDB this is done a bit differently. One to one relationships should be directly embedded. Because the user only has one name, this should be in the user. For one to few relationships they should also be embedded (Karlsson, 2020), like the screenshots a package has. Another one to many relationship the Ivanti App Market has is between the creator and the package. One creator can have multiple packages, but a package only has one creator. This can be stored by giving the creator an array of application_id's, this way it is possible to reference the application. The relationship between the packages and the customers who have downloaded the packages is a many to many relationship, a customer can download multiple packages and one package can be downloaded by multiple customers. One way of storing this would be giving both the package and customer an array of ids to reference this. However, because one package can be downloaded by thousands, maybe even millions of customers it would not be a good idea to give the application an array of customer_id's. This can be fixed by only giving the customer an array of application_id's.

If an object is needed on its own, it is better not to embed it, but join it with another schema.

2.4 How to read the information

Research Method: Literature Study, Problem Analysis

To access the data that is stored in MongoDB in the application, it needs to be connected with Spring Boot. This can be done using the MongoTemplate class and the MongoRepository interface. With the MongoTemplate class it is possible to have more control over your queries, the MongoRepository is more convenient to use, because the basic queries are already defined (devs5003, 2021).

Some POJOs (Plain Old Java Objects) need to be defined first (MongoDB, n.d.). The most important ones are for the application and the user. Some other POJOs that are needed are for the review and the responses, the downloads and the ratings. Another important POJO is for the versions. The application POJO contains all the information that is related to the package, like the name and description. This is what is shown on the website when a customer wants to view the application. The version POJO contains the package itself.

A Repository class needs to be created in order to perform CRUD operations. This repository extends from the MongoRepository interface, making all the CRUD methods accessible. When a parameter is required by a specific query, for example for the FindApplicationByName method a “name” is needed, the method has to be annotated with @Query.

The @Query annotation is applied at the method-level in MongoRepository interfaces, and pertains to a single method (Tegmark, 2022). The @Query annotation has many different elements that can be used:

- Value: this expects a JSON format, inside the value the parameters can be specified. The data will be fetched for the given value, for example findByID or existsByName.

3. Conclusion

A lot of data needs to be stored for this project. The packages need to be stored, but there is also a lot of data that is directly linked to the applications. For this project we have three documents, one that contains all user data, one that contains all application data and one that contains all review information. By using a non-relational database we need less tables and don’t have as many joins, to join all of these tables. This means that it should be faster to load data.

References

devs5003. (2021, May 7). *Spring Boot MongoDB Using MongoTemplate Examples*. Making Java Easy To Learn. Retrieved May 23, 2022, from https://javatechonline.com/spring-boot-mongodb-using-mongotemplate-examples/

Guide, S. (2021, September 14). *MongoDB - Embedded Documents*. GeeksforGeeks. Retrieved May 19, 2022, from https://www.geeksforgeeks.org/mongodb-embedded-documents/

Karlsson, J. (2020, December 4). *MongoDB Schema Design Best Practices*. MongoDB. Retrieved May 19, 2022, from https://www.mongodb.com/developer/article/mongodb-schema-design-best-practices/

MongoDB. (n.d.). *Spring Boot Integration With MongoDB Tutorial*. MongoDB. Retrieved May 23, 2022, from https://www.mongodb.com/compatibility/spring-boot

MongoDB. (2021, July 13). *Database References — MongoDB Manual*. MongoDB. Retrieved May 19, 2022, from https://www.mongodb.com/docs/manual/reference/database-references/

Tegmark, M. (2022, January 12). *Spring Data MongoDB - Guide to the @Query Annotation*. Stack Abuse. Retrieved May 23, 2022, from https://stackabuse.com/spring-data-mongodb-guide-to-the-query-annotation/