**Web programming (server side) – Final project**

Elliot Palierne

337745269

Maor Massas

314801887

**Abstract:**

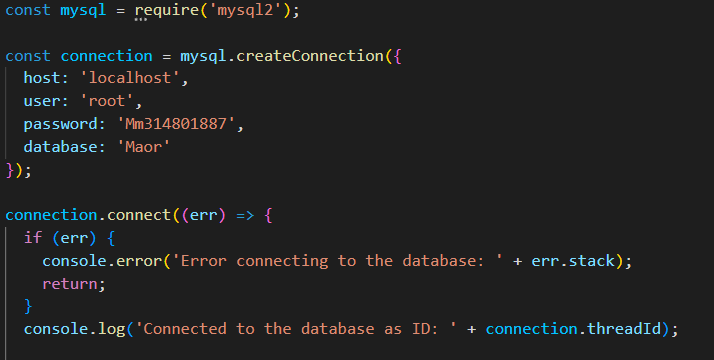
This research project aims to analyze and evaluate the Sequelize library which is an open source Node.js library that allows interaction between JavaScript codes and SQL databases. In order to analyze the Sequelize library we first showed what are the other libraries used in this field such as the MySQL and the TypeORM libraries. Then in the paper review part we explained how Sequelize works, what problems it aims to solve and the relevance of its usage. The research involved running Sequelize within a Node.js application in Visual Studio Code. A systematic methodology was used to achieve the research objective. The methodology involved defining models in JavaScript code, which were then translated into SQL tables by Sequelize. In addition, the study examined Sequelize's built-in commands such as showing all the data, using “raw” SQL queries and also its compatibility with multiple databases. We concluded that Sequelize should be used only in projects involving big and complex databases.

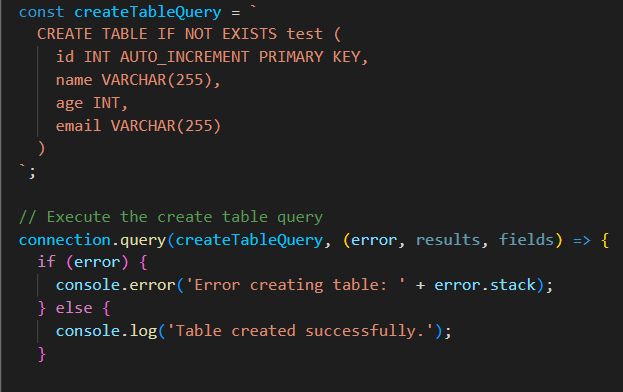
**Introduction:**

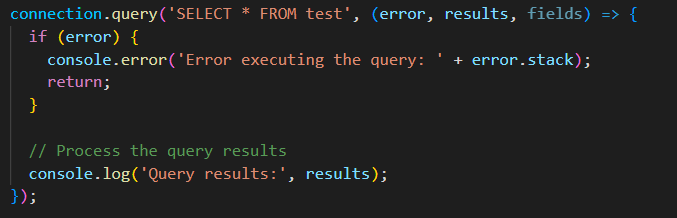
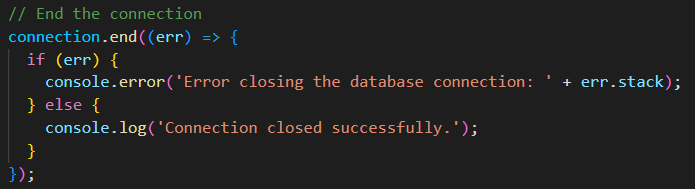
The purpose of this course's final project is to find and analyze an open source Node.Js library that uses a web server and “Sequelize” is such a library. Sequelize is a promise-based ORM (Object–relational mapping) using Node.js, meaning that it works through sending transactions in JavaScript to an SQL database. The purpose of “Sequelize” is to simplify the interaction between JavaScript code and relational databases such as MySQL, PostgreSQL, SQLite, and others. “Sequelize” facilitates the work of developers working with databases thanks to the fact that it “translates” databases tables to JavaScript objects and vice versa. The main advantage of this “translation” is that it eliminates the need to write raw SQL queries, allowing developers to work with databases using JavaScript methods and syntax only. Another advantage of the “Sequelize” library is that it can work with all the SQL programs in contrast to other ORM libraries.

**Related Work:**

In general ORM libraries work with a specific database software. For example this is how a developer using JavaScript works with the popular MySQL database :

*Connection process*

*Creating a table*

*Using a query**Ending the connection*

As shown here is the MySQL official library and its usage. It uses both JavaScript codes and SQL commands thus requiring developers to have knowledge of both of these languages in order to work with SQL databases.

Unlike this library, there are ORM libraries (other than “Sequelize”) that use only JavaScript or TypeScript codes to communicate with the MySQL database. Such libraries are “TypeORM”, “Bookshelf” and “Waterline”. For example here is an example of how to use the “TypeORM” library:

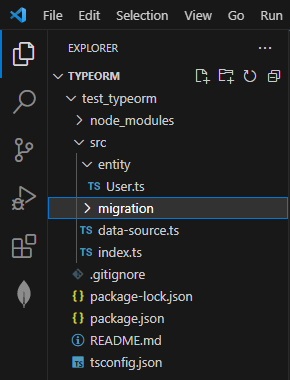
First, you need to download through NPM the “TypeORM” library” by typing this in the VS code terminal:



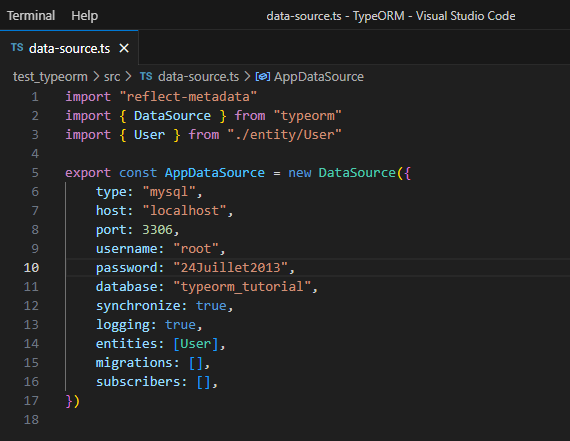
Then when downloaded, you can type in the terminal the following command that will create a new “TypeORM” project:



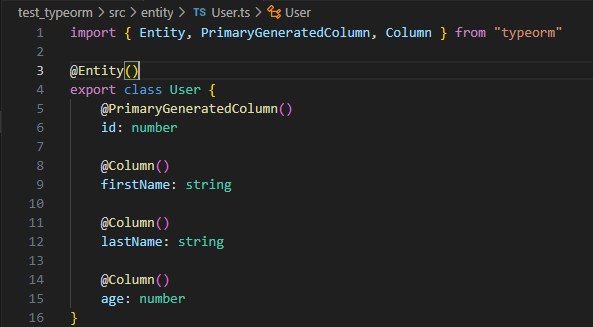
After the ”--name” part, you type the name of your project and after the “--database” part, you type which database you want to use. In this case we will use “MySQL”. After this command a few files will be created in your folder allowing you to use “TypeORM”:



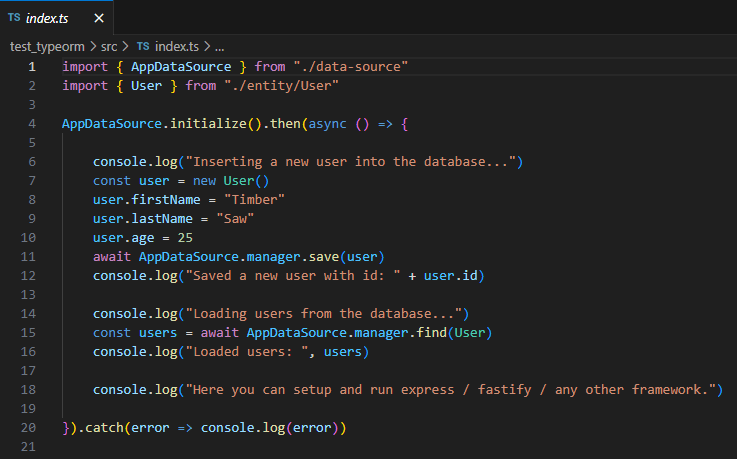
The datasource.ts is the file creating a datasource class that gets exported to the index.ts file. The purpose of this class is to create a connection instance with all the credentials needed to connect to the database server.



Besides the credentials, the DataSource object also contains the “synchronize” and the “logging” attributes. The first one is to allow the “translation” of “TypeORM” entities to SQL tables and the second one is to send to the VS terminal all the SQL messages. The “entities” attribute is an array that contains all the entities that should be sent to the SQL server. Here is an example of an “TypeORM” entity:



As shown here, this is a “TypeORM” entity that will be translated into an SQL table. It uses decorators to explain what the columns are, for example the “@PrimaryGenratedColumn” is the main key of the entity. To send those entities to the server, the script runs the index.ts file that is the main code of the project. For example if a developer wants to create the previous entity shown and add an SQL entity to it, this is what should be written in the index part before its running:



The “TypeORM’ library and the MySQL library are some of the most famous libraries used by developers nowaday in the field of SQL databases.

**Paper Review:**

Problem Definition:

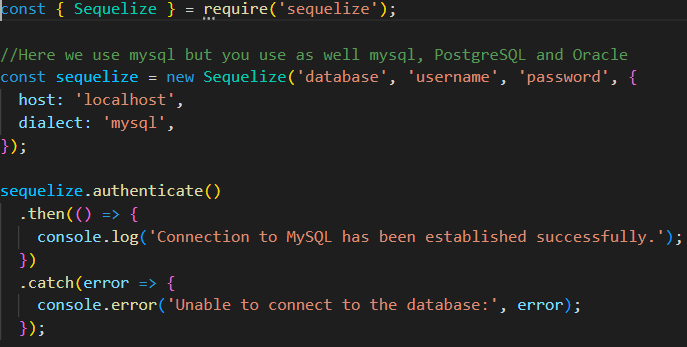
The problem that Sequelize aims to solve is the complexity of working with databases directly in Node.js. By providing a simple and intuitive API for defining models and querying data, Sequelize allows developers to focus on building their application logic rather than worrying about low-level database details. Another problem that “Sequelize” tries to solve is the complexity of sending data from a database to another. At last, when it comes to complex developing projects with huge databases, the data can change rapidly thus forcing the SQL queries to be changed regularly and “Sequelize” solves that through its built-in queries commands written in JavaScript. Thanks to these commands, “Sequelize” also solves the problem that some developers prefer not to learn or use the SQL language due to the fact that it can be considered complex.

Novelty:

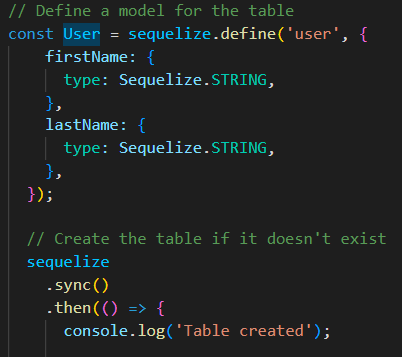
While Sequelize is not necessarily a novel concept (ORMs have been around for many years), it has gained popularity in recent years due to its ease of use and support for multiple databases. Additionally, Sequelize provides advanced features such as associations between tables and transactions that make it a powerful tool for building complex applications.

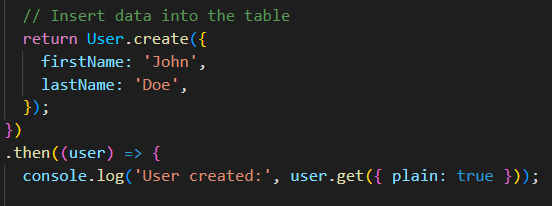
Methodology:

To perform the research, we ran the Sequelize library using the node.js application in Visual Studio Code. We created a connection between the JavaScript code and a SQL database hosted in our computer with the MySQL software. It is important to note that even though we used the MySQL software, MMSQL, PostgresSQL, Oracle and many other SQL softwares can be used with Sequelize.

*Connection Code*

“Sequelize” works through a system of models written in JavaScript codes. Those models are later translated to an SQL table. Next, we executed various test cases to assess the capabilities of Sequelize. These test cases encompassed essential operations such as creating tables, updating data, reading data, and deleting data. For instance, we created models with specific attributes using the Sequelize define function and synchronized them with the database using the sequelize.sync() method. We examined the generated SQL queries and ensured that the tables were created or modified accordingly. In addition, use other ORM literature such as TypeORM mysql in order to draw conclusions about the capabilities and advantages of using Sequelize as an ORM tool for working with SQL databases in Node.js. Here is an example of creating a model:

*Creating the model*

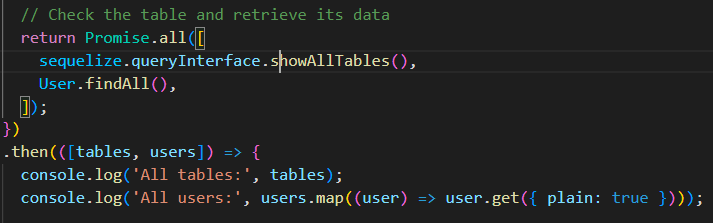
*Filling data into the model*

The sequelize.define function is used to define the “User” model. The first argument is the name of the model, which is “user” in this case. The second argument is an object that defines the model's attributes. The “firstName” and “lastName” attributes are defined with the type Sequelize.STRING, which means it will store string values. When creating a table through a model, “Sequelize” provides two default settings for the new table that can be disabled with the line “DISABLE\_SEQUELIZE\_DEFAULTS” being added at the end of the model’s block code. These two settings are “freezing” the name of the table and adding time stamps. These time stamps are two fields added to the SQL table which are “createdAt” and “updatedAt”. As said in their names, the first field contains the time when the instance was created and the second one the time of the latest update of the instance.

After defining the “User” model, the sequelize.sync() method is called and synchronizes the defined models with the database thus creating the tables if they don't already exist. Once the table is created the .then() method is called causing the message “Table created” to be printed on the log console being in our case the Visual Studio Code terminal.

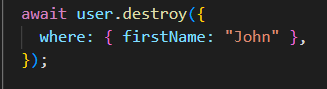
After creating the model “user” and making it an SQL table on our SQL server (located on a local host in “MySQL” in our case), we then want to insert data into the “user” table using the User.create() method. Using this method, it creates a new user instance with the firstName field being “John” and the lastName field being “Doe”. After the creation of the new “user”, the desynchronized method “.then()” is used and it prints the message: ”user created:”. The user.get({ plain: true }) method is used to retrieve the plain JavaScript object representation of the user without additional Sequelize metadata. As shown here, all these steps were solely written in JavaScript and without any SQL thus allowing developers to use SQL databases without knowledge of the SQL language.

Besides these commands shown previously, “Sequelize” also has built-in commands to communicate with the SQL server such as “showAllTables”, “destroy” or ”update”. Sequelize also supports advanced features such as associations between tables, transactions, and migrations. As shown here the codes are all written in JavaScript but do SQL commands thus allowing developers to not use the SQL language while communicating with an SQL server.

*Example of use to the showAllTables function:*

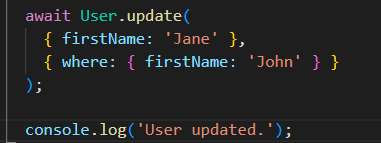
This code retrieves two lists containing all the tables and users of the database by using a promise containing the sequelize.queryInterface.showAllTables() command and the User.findAll() method. Once the promises are successfully used, the code logs prints the lists of the tables and the users to the console.

*Use of destroy function:*



This code deletes records from the "user" table in the database using a criteria. Here it specifically deletes all the records in the table that have “John” in the firstName column using the destroy() method. The await keyword ensures that all the other asynchronized operations are stopped in order to commit the delete promise and also to throw to the console the rejected value in case the promise is rejected.

*Use of update function:*



That code uses Sequelize’s methods to update the firstName column of all records in the "user" table to be filled with the word 'Jane'. After the update operation is complete, it prints a confirmation message ('user update') to the terminal.

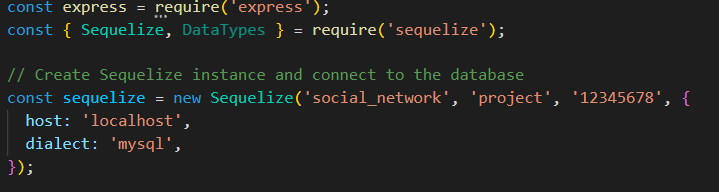
In addition to these built-in commands, it is also possible to use “raw” SQL queries with Sequelize if needed with the sequelize.query(“*Here should be the SQL code*”) method.

**Conclusion:**

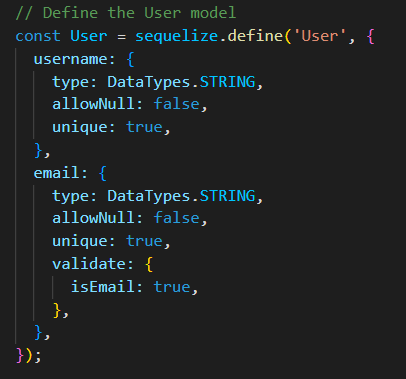
Overall, we found that Sequelize is a powerful tool when there is a need for developers to interact with one or multiple databases which is the case for building web applications for example. Its relative ease of use and support for multiple databases make it an attractive choice for developers who want to focus on building their application logic rather than worrying about low-level database details. Additionally, its advanced features such as associations between tables and transactions make it a versatile tool for building complex applications.

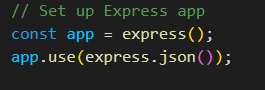
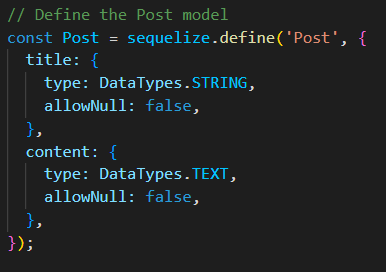
However it is important to notice that while Sequelize is an useful tool to work with complex databases, it seems that the use of Sequelize can be less efficient for projects using low-level databases. The reason for that is due to the fact that Sequelize forces developers to learn a new syntax and also to understand the Sequelize’s entities logic in order to use it. Another reason is that using Sequelize for the first time can be pretty hard for new users regardless of its syntax because of the complexity of setting up Sequelize on a new computer. Due to these reasons there aren't any reasons for developers working with low-level SQL databases to use Sequelize or any other ORM libraries.

In conclusion, Sequelize should be used for projects using complex databases. A good example can be the development of a Social Network application. We will show here a brief code for such a project. In order to achieve this project we also use the “Express” library which is a server framework.

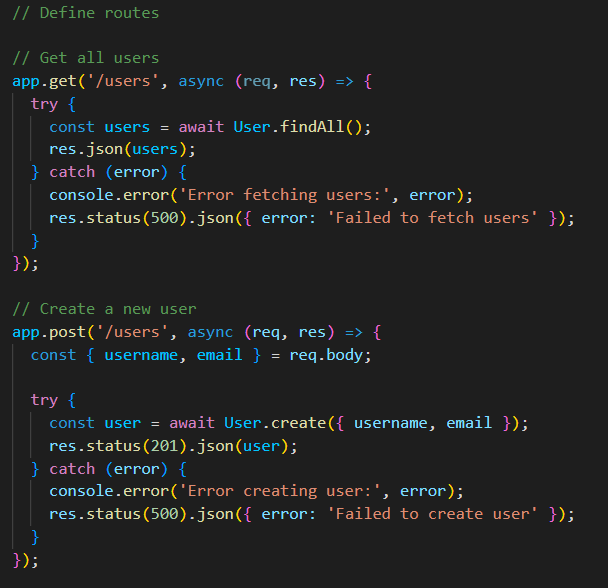
*Connection with the use of express*****

In this code we import the “express” server framework using “require” and also the Sequelize library. After that we also connect to our MySQL server.

****

****

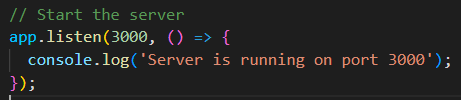
In the last three pictures of code we created two Sequelize entities : “User” and “Post”. We also set up a variable named “app” being the express object. We also sent our data as a JSON code with “app” to our express server.

****

This code retrieves all the users from our database. If successful, the user records are sent as a JSON response, otherwise, an error message is logged and a 500 status code response is sent. Then, the second part of the code is an Express route handler that creates an SQL record through an Sequelize entity using data from the “req” variable which is the data retrieved from the client-side that isn’t shown here. If successful, the code sends a JSON response with the created user and a 201 status code. If an error occurs, it prints an error message to the console and sends a 500 status code.

****

This code here does the same thing as the two previous code snippets but this time with the “posts” records instead of the “user” records. The only difference is that the data retrieved to create a “post” record uses two parts which are the title and the content of the post unlike the creation of an “user” record that needed the email and username for its creation.

****

The code here starts the server and listens for incoming HTTP requests on port 3000. When the server starts successfully, it prints a message to the console indicating that the server is running on port 3000.

**References:**

GitHub-sequelize:

<https://github.com/sequelize/sequelize>

GitHub-typeorm:

<https://github.com/typeorm/typeorm>

npm-mysql2:

<https://www.npmjs.com/package/mysql2>

Sequelize website:

<https://sequelize.org/>

typeORM website:

<https://typeorm.io/>

# Node.js ORMs: Why you shouldn’t use them:

<https://blog.logrocket.com/node-js-orms-why-shouldnt-use/#why-sequelize-first-place>

# How To Use Sequelize with Node.js and MySQL:

<https://www.digitalocean.com/community/tutorials/how-to-use-sequelize-with-node-js-and-mysql>

Internet Israel - Express.js:

<https://internet-israel.com/%D7%9E%D7%93%D7%A8%D7%99%D7%9B%D7%99%D7%9D/express/express-js-%D7%9E%D7%91%D7%95%D7%90/>