node DB

Connecting node to DB using sequelize

Our goals

- Choosing my database
- Connect to database
- What is ORM?
- Solve common db problems
- Understand what is a REST server and create our first REST server

Database types

- There is a large variety of database solutions you can choose so first let's understand the different categories we can choose from
  - Relational DB (MySQL, Postgres, Oracle)
  - NOSQL (MongoDB, Couchbase)
  - Graph Database (Neo4J)
  - Key Value (Redis, MemCached)

Database types

- There is a large variety of database solutions you can choose so first let's understand the different categories we can choose from
  - Relational DB (MySQL, Postgres, Oracle)
  - NOSQL (MongoDB, Couchbase)
  - Graph Database (Neo4J)
  - Key Value (Redis, MemCached)

Postgres

- Postgres is a relational database
- SQL based
- Database is splitted to tables
- Tables are splitted to rows
- Rows are identified by a primary key
- Fast, Mature with a lot of built in features
- In this lesson we will learn to work with postgres

Our first db query

- For this example we will require a running database postgres server
- Can be easily opened using AWS (eligible for free tier)
- Make sure your database is publicly open
  - You should have a database endpoint like:
    - ****.****.eu-west-1.rds.amazonaws.com
- When creating a database you were asked to provide a username and a password
- Also you had to provide a port
- From all these data you can create a database url:
  - postgres://username:password@ ****.****.eu-west-1.rds.amazonaws.com:5432

Our first db query - EX

- After opening our database, we will now do the following
- Connect to the database using node
- Create a new table called tasks which will contain the following columns
  - id - the primary key number
  - title - a string of max length 100
  - description - an unlimited string
  - when - date and time
- we will insert rows to that table
- node has a postgres driver that can connect and execute queries, pg

Our first db query - run the following EX

```
CREATE TABLE tasks(id INTEGER PRIMARY KEY,title VARCHAR
(50),description TEXT, time TIMESTAMP)

INSERT INTO tasks VALUES(1, 'task1', 'Buy soya milk', '2019-4-2
10:50');

INSERT INTO tasks VALUES(2, 'task2', 'Buy tofu', '2019-4-2 10:51');

INSERT INTO tasks VALUES(3, 'task3', 'Buy computer', '2019-4-2 10:52');

INSERT INTO tasks VALUES(4, 'task4', ‘Walk Piglet', '2019-4-2 10:53');

INSERT INTO tasks VALUES(5, 'task5', ‘Take piglet to the park',
'2019-4-2 10:54’);
```

ORM - Object Relational Mapping

- We have tables in database
- We need to query information from those tables within our js code
- We use OOP in our code it will be nice to get the data not as rows but as instance of classes
- ORM maps data to OOP
- In our case we aim to map our database data to OOP class instances
- ORM hides the database engine so we don’t need to worry about query syntax, and we can easily change our database engine

Sequelize

- ORM for relational databases
  - Postgres, MySQL, MariaDB, SQLite, Microsoft SQL Server
- Promise based
- You create instance by providing the database url and sequelize will connect to the database
- We define our ORM models
- We query by using those models

Sequelize - Connecting to our db - ex

- Our first ex will be to connect to our database using sequelize
- We do that by creating an instance of Sequelize and providing the database url to the constructor
- Sequelize will use the strategy pattern to connect to different db's so it will look at the database url and will know automatically to choose the right strategy
- Each db strategy requires a driver to connect to the database
- In our example we are connecting to postgres db so we need to install the pg driver we installed earlier
- postgres support hstore data type for storing key value in a single column so to support this feature we will also need to install pg-hstore
- Let's connect to our db using sequelize and use the authenticate method to verify that we are connected.

Sequelize - Task model ex

- Our job now is to create our first model
- That model should map the table we created earlier
- We will use our model to query our database and place the data in instances of our model
- Our models will usually be placed in separate files
- We will use this module for CRUD operations

Sequelize - Table relations

- Tables can be connected to each other for example
  - we might create a user table with row for each user in our system
  - Each user can create many tasks so each row of the task table will point to the user it belongs to
- Different kind of table relations
  - 1:1
  - 1:M
  - N:M

- Let’s go over those relations and learn how to deal with them in our database and using sequelize

1:1 - ex

- one table is referencing another table
- We will create 2 tables: users, settings
- each user has a single row in the settings table that represents his application settings
- with 1:1 one of the tables contains a column for example settingId that is referencing the primary key from the other table
- If we have additional data about the connection for example each connection is either default or customized then we can place that data in one of our tables, important to note that in that case it is common that the column describing the connections should be in the same table
- Create those tables and relation with sequelize along with the connection extra data

1:M - EX

- This relation is when a row in tableA can reference many rows in tableB
- Each row in tableB will have a column that points to the pk of tableA that he is connected to
- If connection contains additional data it will be presented in tableB
- in our example let’s connect between the user table and the task table
- each user can have many tasks
- each task will contain priority value specifying data on the connection on how strong the priority of this task is to the user

N:M - EX

- This relation is when Each row in TableA can reference many rows in tableB and each row in tableB can reference many rows in table A
- This relation is done via a middle table referred as through table
- Each row in the through table contains a column of pk of a row from TableA and a column for pk of a row from TableB
- Each row of the through table represents a single connection
- The through table can contain more columns of data about a single connection
- Create a Tags table contains a single column for title
- Create a N:M connection between the tags and the tasks
- the Through table should contain an order column of number to help arrange the tags by order
- Query tags of a single task and order it with the through order column

Generic foreign key / Polymorphic association

- Generic foreign key is used when we want to reference several tables
- For example a tag in our application can tag a user or a task
- We can solve this case if we create a generic foreign key where we have an id of a tag on one columns and on another column we have an id of the table we are referencing plus the table name in another column
- Extend the Many to Many exercise we did before and this time our tags can reference a user as well
- try and query users by tags and tasks by tags.

Validation - EX

- Validations are defined on the sequelize model
- Validation can be on a certain field or on the entire model
- You can also create custom validation
- Create a user model that contains the following fields
  - email, password, repeatPassword, hashedPassword
- The password and repeat has to match
- Create validation for those fields
- Create also a custom validations
- make sure the original password is not saved only the hashed one
  - use the bcrypt package
- Create an instance method that will authenticate the password

Indexing

- Indexes are special lookup tables that we create to speed data retrieval
- index tables speed SELECT and WHERE
- Index slow down INSERT, UPDATE
- We can create or drop an index with no effect on the data
- We can index single column or multiple columns in the same index table
- In postgres index for primary key is created automatically
- When index should be avoided
  - small tables
  - tables with plenty of updates and inserts
  - on columns that contain a lot of null values or change a lot

Indexing- EX

- On the user table we create let’s create an index for the email field

Migrations

- We will often need to change the structure of the tables in our database
- Changes to tables are called migrations and they need to be defined declaratively
- migrations describe how to get to the new state of a table, and how to revert to previous state
- 2 types of migrations
  - data / seed migration - where we insert data to the table
  - structure migration
- Another table will be created in the database saving the migration history

Sequelize CLI - EX

- The cli will bootstrap sequelize in our app
  - config
  - models
  - migration
- install with npm
  - npm install sequelize-cli —save-dev
- init sequelize using the cli
- create a user model
- add migration to add a column to that table
- add migration to populate that table with seed data

REST server

- REST is a communication protocol based on HTTP
- it uses the request to understand from which table the data is located and what we want to do with that data
- The url will determine the table
- the request type will determine what we want to do with the data
- Use express app generator to start a new app and build a rest api to the users table

Summary

- In this lesson we learned best practices with our database and how to manage everything with sequelize
- We learned how to quickly create a REST server connected to a database