Getting Started

1. **Mongo Installation**: First, we need MongoDB to be installed in our machine (check the MongoDB Super Skill if needed)

Let’s have a look at some of the basics of Mongoose by implementing a model that presents the data of a simplified address book.

Fire up your favorite IDE (ahem..VSCode), create a blank project, and let’s get started! We will be using the limited ES6 syntax in Node, (so we won’t be configuring Babel).

2. **Mongoose Installation**

   Let’s go to the project folder and initialize our project:

```
npm init -y
```

Let’s install Mongoose and a validation library with the following command:

```
npm install mongoose validator
```

The above install command will install the latest version of the libraries. *The Mongoose syntax in this course is specific to Mongoose v5 and beyond.*

Getting Started

3. **Database Connection**

   Create a file `./src/database.js` under the project's root.

   Next, we will add a simple class with a method that connects to the database.

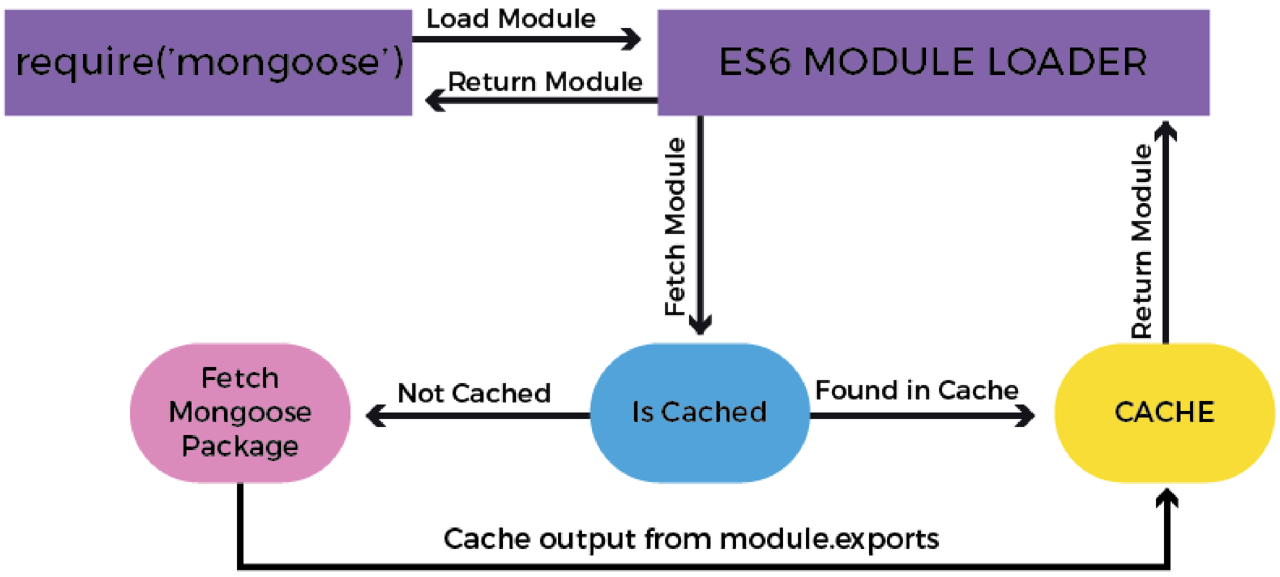
   Your connection string will vary based on your installation.

```
let mongoose = require('mongoose');
const server = '127.0.0.1:27017'; // REPLACE WITH YOUR DB SERVER
const database = 'myDB';      // REPLACE WITH YOUR DB NAME
class Database {
  constructor() {
      this._connect()
    }
  _connect() {
      mongoose.connect(`mongodb://${server}/${database}`)
        .then(() => {
          console.log('Database connection successful')
        })
        .catch(err => {
          console.error('Database connection error')
        })
    }
  }
  module.exports = new Database()
```

The `require('mongoose')` call above returns a Singleton object. It means that when you call `require('mongoose')` for the first time, it creates an instance of the Mongoose class and returns it. On subsequent calls, it will return the same instance that was created and returned to you the first time because of how module import/export works in ES6.

Side Note

About Module import/require work-flow:



In the previous code, we have also turned our database class into a singleton by returning an instance of the class in the `module.exports` statement because we only need a single connection to the database.

ES6 makes it very easy for us to create a singleton (single instance) pattern because of how the module loader works by caching the response of a previously imported file.

‹ Previous next ›