

Objective

In this checkpoint, you will be performing a series of guided instructions to handle and manage your database while using Mongoose

- **PS:** don't forget to comment on your code before submitting it.

Instructions

Installing and setting up Mongoose:

Add MongoDB and Mongoose to the project's package.json. Store your MongoDB Atlas database URI in the private .env file as MONGO_URI. Surround the URI with single or double quotes and make sure no space exists between both the variable and the '=' and the value and '='. Connect to the database using the following syntax:

```
`mongoose.connect(<Your URI>, { useNewUrlParser: true, useUnifiedTopology: true });`
```

Create a person with this prototype:

- Person Prototype -

name: string [required]

age: number

favoriteFoods: array of strings (*)

Use the mongoose basic schema types. If you want you can also add more fields, use simple validators like required or unique, and set default values. See the [mongoose docs](#).

Create and Save a Record of a Model:

Create a document instance using the Person constructor you built before. Pass to the constructor an object having the fields name, age, and favoriteFoods. Their types must conform to the ones in the Person Schema. Then call the method document.save() on the returned document instance. Pass to it a callback using the Node convention.

```
/* Example */  
  
// ...  
  
person.save(function(err, data) {  
  
  // ...do your stuff here...  
  
});
```

Create Many Records with model.create()

Sometimes you need to create many instances of your models, e.g. when seeding a database with initial data. Model.create() takes an array of objects like [{name: 'John', ...}, {...}, ...] as the first argument and saves them all in the DB.

Create several people with **Model.create()**, using the function argument arrayOfPeople.

Use model.find() to Search Your Database

Find all the people having a given name, using Model.find() -> [Person]

Use model.findOne() to Return a Single Matching Document from Your Database

Find just one person which has a certain food in the person's favorites, using Model.findOne() -> Person. Use the function argument food as a search key.

Use model.findById() to Search Your Database By _id

Find the (only!!) person having a given _id, using Model.findById() -> Person. Use the function argument personId as the search key.

Perform Classic Updates by Running Find, Edit, then Save

Find a person by _id (use any of the above methods) with the parameter personId as a search key. Add "hamburger" to the list of the person's favoriteFoods (you can use Array.push()). Then - inside the find callback - save() the updated Person.

Note: This may seem tricky, if, in your Schema, you declared favoriteFoods as an Array, without specifying the type (i.e. [String]). In that case, favoriteFoods defaults to Mixed type, and you have to manually mark it as edited using document.markModified('edited-field'). See [Mongoose documentation](#)

Perform New Updates on a Document Using model.findOneAndUpdate()

Find a person by Name and set the person's age to 20. Use the function parameter personName as a search key.

Note: You should return the updated document. To do that you need to pass the options document { new: true } as the 3rd argument to findOneAndUpdate(). By default, these methods return the unmodified object.

Delete One Document Using model.findByIdAndRemove

Delete one person by the person's _id. You should use one of the methods findByIdAndRemove() or findOneAndRemove(). They are like the previous update methods. They pass the removed document to the DB. As usual, use the function argument personId as the search key.


MongoDB and Mongoose - Delete Many Documents with model.remove()

Delete all the people whose name is "Mary", using Model.remove(). Pass it to a query document with the name field set, and of course, do a callback.

Note: The Model.remove() doesn't return the deleted document, but a JSON object containing the outcome of the operation, and the number of items affected. Don't forget to pass it to the done() callback, since we use it in tests.

Chain Search Query Helpers to Narrow Search Results

Find people who like burritos. Sort them by name, limit the results to two documents, and hide their age. Chain .find(), .sort(), .limit(), .select(), and then .exec(). Pass the done(err, data) callback to exec().

 Learn by making.

Powered by GO MY CODE
[Privacy Policy](#)