MongoDB and Promises

We will start the task manager project
We need authentication system.

Go to MongoDB.comMany nodes developers use MongoDB. You can also use mysql or Postgres as well. Mongodb is a no sql database unlike mysql and Postgres. There are tables in sql databases. In nosql databases there are collections instead of tables. In tables, there are rows or records for list of entries and also columns for different parts of the data. In nosql databases there are documents instead of rows and fields instead of columns.

For mac and linux, installation:
Click get mongodb button and select the server tab. We are going to use community server. Version is the current release. OS is your OS version. For linux, choose the correct distribution. You can select TGZ for the package. Double click the archive and go to the bin directory. We can move the bin directory to a permanent location (user directory) after updating the name to mongodb. Create a data directory on the hard drive. Create a new folder named mongodb-data at the same directory with mongodb to store the data.

Go to vs code terminal go to the bin directory for the mongodb. Run ~/mongodb/bin/mongod —dbpath=~/mongodb-data
We can see that the server is up and running.

Check the port number on the displayed message to see the default port where mongodb runs.

Windows installation:
Click getmongodb button. Select the server tab. Get community server. Select current version, your os version and select zip for package. Click the download button. Extract to downloads folder. Open the folder and in the bin directory there are executables. Rename the folder to mongodb. Then move it to user directory (C:Users/Can/). At the same directory create a new folder named mongodb-data. Mongodb has executables and mongodb-data will store the collections. In the vs code terminal go to the mongodb directory and run mongodb/bin/mongod.exe —dpath=/Users/Can/mongodb-data
We can check the port number in the given output message.

We should leave this terminal up and running to be able to access the database.

Installing GUI viewer: database admin tool
Google robo 3T and install robo 3T
Click the download button and select your os. Open robots 3t after installation.
Select “create” to create a connection. We will change the name under connection: Local MongoDB Database. The rest is same. Click the test button. Then click the save button. Open the connection.
Right click the connection and select new shell. Write db.version() and click run button to see the result.

Connecting and inserting documents
We are going to use the native driver to connect to mongodb.
Go to mongodb.com and open the documentation, mongodb drivers.
Find nodes and click it. Check releases and click on the API.

Google npm mongodb and check the first result

Go to the vs code and open a new terminal. Do not close the mongodb terminal.
Create a new folder named task-manager. Go to the new folder using cd from terminal.
Run npm init -y to initialize npm.

Install mongodb module: npm i mongodb

Create a new file named mongodb.js
In the new file we will implement CRUD operations: create, read, update and delete.

```
const mongodb = require('mongodb')
const MongoClient = mongodb.MongoClient

const connectionURL = 'mongodb://127.0.0.1:27017'
const databaseName = 'task-manager'

MongoClient.connect(connectionURL, { useNewUrlParser: true}, (error, client) => {
        //options object and a callback function to be run after connection

        if (error) {
                return console.log('Unable to connect to database')
        }
        console.log('Connected correctly')
})
```

Go to terminal and run node mongodb.js

Check the other terminal to see the new connection attempt. There is a connection pool mechanism and multiple connections can be created. CTL C to break.

Now change it to..

```
MongoClient.connect(connectionURL, { useNewUrlParser: true}, (error, client) => {
        //options object and a callback function to be run after connection

        if (error) {
                return console.log('Unable to connect to database')
        }
        const db = client.db(databaseName)
        db.collection('users').insertOne({
                name: 'Can',
                age: 20
        })
})
```

Save and run the script. node mongodb.js
Check if the document is inserted to the collection.
Open robo 3t

Right click the connection and choose refresh. We can see collections and users document. Right click the document and select view to view the data. Id is a unique identifier created by mongodb.

Inserting documents

Insertone is an asynchronous function.

```
MongoClient.connect(connectionURL, { useNewUrlParser: true}, (error, client) => {
	//options object and a callback function to be run after connection

	if (error) {
		return console.log('Unable to connect to database')
	}
	const db = client.db(databaseName)
	db.collection('users').insertOne({
		name: 'Can',
		age: 20
	}, (error, result) => {
		if (error) {
			return console.log('Unable to insert user')
		}

		console.log(result.ops)
	})
})
```

Save the file and run node mongodb.js
The output is result.ops

Check mongodb docs for collection, insertion, callback, opresult, etc.

Comment insertOne and add the following:

```
db.collection('users').insertMany([
	{
		name: 'Can',
		age: 20
	}, {
		name: 'Cem',
		age: 25
	}
], (error, result) => {
	if (error) {
		return console.log('Unable to insert documents')
	}

	console.log(result.ops)
})
```

Check rob 3t and refresh page to see current documents.

Run node mongodb.js
Check the output and then also robocalls 3t documents.

Comment insertMany, add the following

```
db.collection('tasks').insertMany([
	{
		description: 'Clean the house',
		completed: true
	}, {
		description: 'Water plants',
		completed: false
	}, {
		description: 'See the doctor',
		completed: false
	}
], (error, result) => {
	if (error) {
		return console.log('Unable to insert tasks')
	}

	console.log(result.ops)
})
```

Run node mongodb.js and check the output.

Object ID
IDs are GUID, global unique identifiers.

Add this at the top

```
const ObjectID = mongodb.ObjectID
```

Then comment all above and use this instead

```
const { MongoClient, ObjectID } = require('mongodb')
```

Generate our own ids

```
const id = new ObjectID()
console.log(id)
```

Run node mongodb.js
Check the id.
Ctl C to shut down the connection.

```
const id = new ObjectID()
console.log(id)
console.log(id.getTimestamp())
```

Rerun the code.

Rerun the code. Compare the timestamps.

```
	db.collection('users').insertOne({
		_id: id,
		name: 'Tansel',
		age: 26
	}, (error, result) => {
		if (error) {
			return console.log('Unable to insert user')
		}

		console.log(result.ops)
	})
```

Rerun the code.
Check users ids from robots 3t

```
const id = new ObjectID()
console.log(id.id)
```

To see the raw id in binary.

```
const id = new ObjectID()
console.log(id.id.length)
```

Size is 12 bytes

```
const id = new ObjectID()
console.log(id.id.length)
console.log(id.toHexString.length)
```

Size is 24 bytes now.

Querying documents

Remove object id related stuff from the code.
Also remove insert codes.

```
MongoClient.connect(connectionURL, { useNewUrlParser: true}, (error, client) => {
	//options object and a callback function to be run after connection

	if (error) {
		return console.log('Unable to connect to database')
	}
	const db = client.db(databaseName)

	db.collection('users').findOne({ name: 'Can' }, (error, user) => {
		if (error) {
			return console.log('Unable to find user')
		}
		console.log(user)
	})
})
```

Save and rerun. Check the output.

Try this search

```
	db.collection('users').findOne({ name: 'Can', age: 1 }, (error, user) => {
		if (error) {
			return console.log('Unable to find user')
		}
		console.log(user)
	})
```

Returns null.
If there are two matches, it returns the first one.

If search by object id, use this: _id: new ObjectID("copy string id here")

Ensure 3 users are 26.
There is no callback for find. And it return cursor. Check the docs for details.

```
	db.collection('users').find({ age: 26 }).toArray((error, users) => {
		console.log(users)
	})
```

Comment findOne.
Save and rerun.

```
	db.collection('users').find({ age: 26 }).counts((error, count) => {
		console.log(count)
	})
```

Save and rerun.

Get the last task by its id and incomplete tasks.

```
	db.collection('tasks').findOne({ _id: new ObjectID("the last task") }, (error, task) => {
		if (error) {
			return console.log('Unable to find user')
		}
		console.log(task)
	})

	db.collection('tasks').find({ completed: false }).toArray((error, tasks) => {
		console.log(tasks)
	})
```