SIT 725 Prac 4 Databases

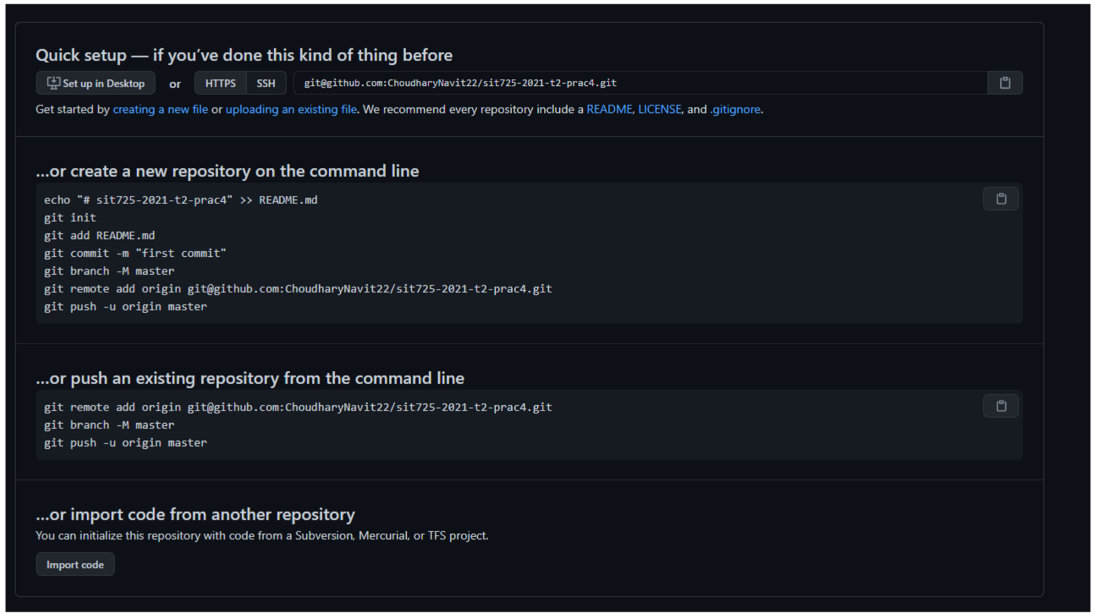
Contents

Creating Week 4 Repository

Welcome to week 4 and this prac we will try to learn how to modify our last week's work with a proper backend server and database.

The first part of this week's prac is to create a repo but this time we won't add README.md file or a license or a .gitignore file. This time we will import the code from previous weeks prac.

We use the import code button at the bottom and add the link of our previous weeks repo to import code to our this weeks repo.



Quick setup — if you've done this kind of thing before

```
git@github.com:ChoudharyNavit22/sit725-2021-t2-prac4.git
```

Get started by creating a new file or uploading an existing file. We recommend every repository include a README, LICENSE, and .gitignore.

...or create a new repository on the command line

```
echo "# sit725-2021-t2-prac4" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M master
git remote add origin git@github.com:ChoudharyNavit22/sit725-2021-t2-prac4.git
git push -u origin master
```

...or push an existing repository from the command line

```
git remote add origin git@github.com:ChoudharyNavit22/sit725-2021-t2-prac4.git
git branch -M master
git push -u origin master
```

...or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

Updating repository for week 4

Next we would want to update things in our repository so that it looks like week 4 and not week 3.

We first update our README.md file and make it to look like week 4, secondly we go into our package.json and update the title and description to be compatible with week 4.

You can also update your index.html and make it to look like for week 4 but that is not a compulsory thing as it won't affect the github maintenance for our repository

Now we are ready to make our first commit for week 4 as we want these changes to be reflected on out github also.

Serving data through REST API’s

Last week when we added cards to our html page we used a constant array of objects to create dynamic cards and we discussed that in real life scenario we would be sending this data from a backend server and then use it in our application to display it to the user.

So let's try to do the same by creating a GET Request on our NodeJs server and connect it to our html page and see if that gives us the same result.

First we modify our server.js file and add a rest api to it. So now our server.js file looks something like this:

Serving data through REST API’s Cont ...

```
var express = require("express")
var app = express()

app.use(express.static(__dirname+'/public'))
app.use(express.json());
app.use(express.urlencoded({ extended: false }));

const cardList = [
    {
        title: "Kitten 2",
        image: "images/kitten-2.jpg",
        link: "About Kitten 2",
        desciption: "Demo desciption about kitten 2"
    },
    {
        title: "Kitten 3",
        image: "images/kitten-3.jpg",
        link: "About Kitten 3",
        desciption: "Demo desciption about kitten 3"
    }
]

app.get('/api/projects',(req,res) => {
    res.json({statusCode: 200, data: cardList, message:"Success"})
})

var port = process.env.port || 3000;

app.listen(port,()=>{
    console.log("App listening to: "+port)
})
```

Serving data through REST API’s Cont ...

Next we modify our scripts.js so that we can make a get request to the server so that we receive our data.

First we add a function in our scripts.js

```
const getProjects = () => {
    $.get('/api/projects',(response) => {
        if(response.statusCode==200){
            addCards(response.data);
        }
    })
}
```

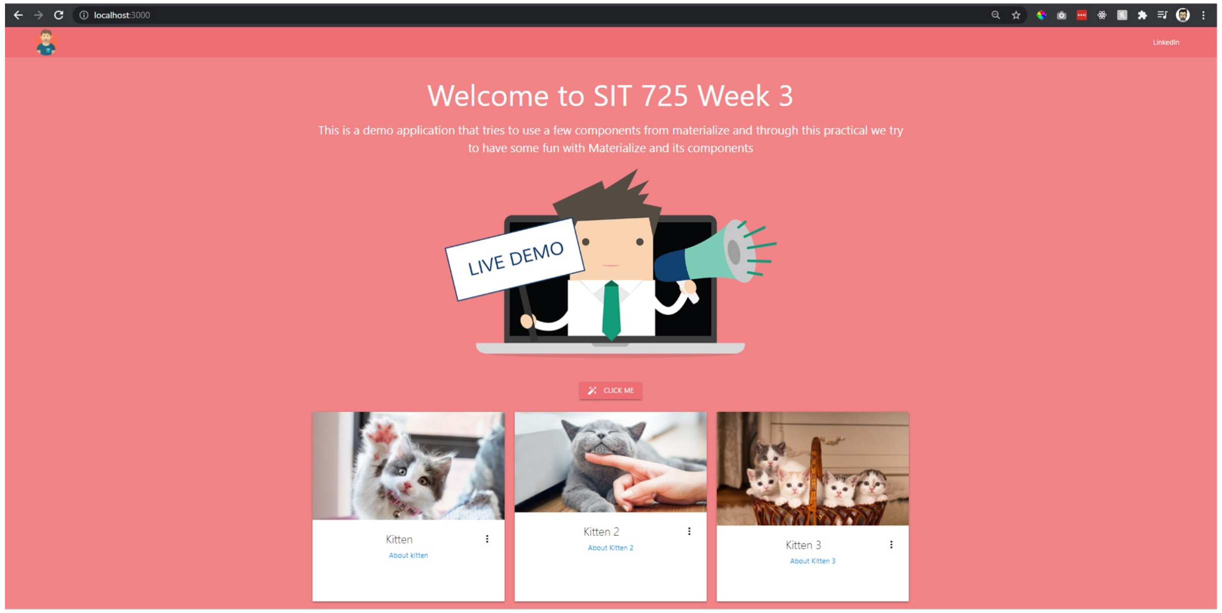
Serving data through REST API’s Cont ...

And next we also modify a section of our scripts.js which is the section we already have written. We modify the $(document).ready section so we can call the function we just created. So now our section looks something like this.

```
$(document).ready(function(){
    $('.materialboxed').materialbox();
    $('#formSubmit').click(()=>{
        submitForm();
    })
    getProjects();
    $('.modal').modal();
  });
```

Serving data through REST API’s Cont ...

Now let's try to run our application and see if everything worked out.



Welcome to SIT 725 Week 3

This is a demo application that tries to use a few components from materialize and through this practical we try to have some fun with Materialize and its components

Serving data through REST API’s Cont ...

Looks exactly the same right. Now that we have done this part we should commit our code so that the modifications we made don't get lost.

In the next part we would learn about NoSQL Database and how we can add and retrieve data from that database.

MongoDB

MongoDB is a document-oriented NoSQL database for storing large amounts of data. It contains the data model, which permits hierarchical connections to be represented. Instead of using tables and rows as in standard relational databases, it employs JSON-like documents with optional schema. The basic units of data in MongoDB are documents that include key-value pairs.

Introduction to MongoDB in Node.js

MongoDB + Node.js = Powerful Backend

- MongoDB stores JSON-like documents (NoSQL DB).
- Node.js can connect using the official mongodb or mongoose library.
- We'll use mongoose for easier data modeling and queries.

Install MongoDB

On Windows

If you haven't already:

Download from https://www.mongodb.com/try/download/community

During installation, check the option to install **MongoDB as a Windows Service**.

Check if it's running > Open **Command Prompt** and type: > `services.msc`

Look for **MongoDB** in the list of services.

Make sure its status is **Running**.

Or run this to test in terminal> `mongo >` If it connects, MongoDB is running.

Install MongoDB

On Mac

Install MongoDB (if needed)

```
brew tap mongodb/brew
brew install mongodb-community
```

Start MongoDB

```
brew services start mongodb/brew/mongodb-community
```

Check if it's running

`brew services list` >> Look for mongodb-community with status started.

* Regardless of OS, you can check with: `mongo` >>if you get the MongoDB shell prompt (>), it means MongoDB is up and running.

Installing Mongoose

In your project folder, run:

```
npm install mongoose
```

Then, import it in `server.js`:

```
const mongoose = require('mongoose');
```

Connecting to MongoDB

```
mongoose.connect('mongodb://localhost:27017/myprojectDB', {
  useNewUrlParser: true,
  useUnifiedTopology: true,
});

mongoose.connection.on('connected', () => {
  console.log('Connected to MongoDB!');
});
```

Goes in server.js file

* Make sure MongoDB is running locally, or use a connection string from MongoDB Atlas.

Creating a Mongoose Schema

```
const ProjectSchema = new mongoose.Schema({
  title: String,
  image: String,
  link: String,
  description: String,
});

const Project = mongoose.model('Project', ProjectSchema);
```

Replacing Static Data with MongoDB Query

Replace your GET /api/projects endpoint with:

```
app.get('/api/projects', async (req, res) => {
  const projects = await Project.find({});
  res.json({ statusCode: 200, data: projects, message: "Success" });
});
```

Example server.js

```
var express = require("express")
var app = express()
var port = process.env.port || 3004
const mongoose = require('mongoose');

// Middleware
app.use(express.static(__dirname + '/public'));
app.use(express.json());
app.use(express.urlencoded({ extended: false }));

mongoose.connect('mongodb://localhost:27017/myprojectDB', {
useNewUrlParser: true,
useUnifiedTopology: true,
});
mongoose.connection.on('connected', () => {
console.log('Connected to MongoDB');
});
// 2. Define your schema and model
const ProjectSchema = new mongoose.Schema({
title: String,
image: String,
link: String,
description: String,
});
const Project = mongoose.model('Project', ProjectSchema);
// 3. REST API route
app.get('/api/projects', async (req, res) => {
const projects = await Project.find({});
res.json({ statusCode: 200, data: projects, message: 'Success' });
});
// 4. Start server
app.listen(port, () => {
console.log(`App listening on port ${port}`);
});
```

Sample Insert Code

To seed/populate the database using a seed.js code:

```
const sampleProject = new Project({
  title: "Kitten 4",
  image: "images/kitten-4.jpg",
  link: "About Kitten 4",
  description: "Demo description about kitten 4"
});

sampleProject.save().then(() => console.log("Sample project
saved!"));
```

How to Check If the Data Was Inserted

Option 1: Use the Mongo Shell

Run

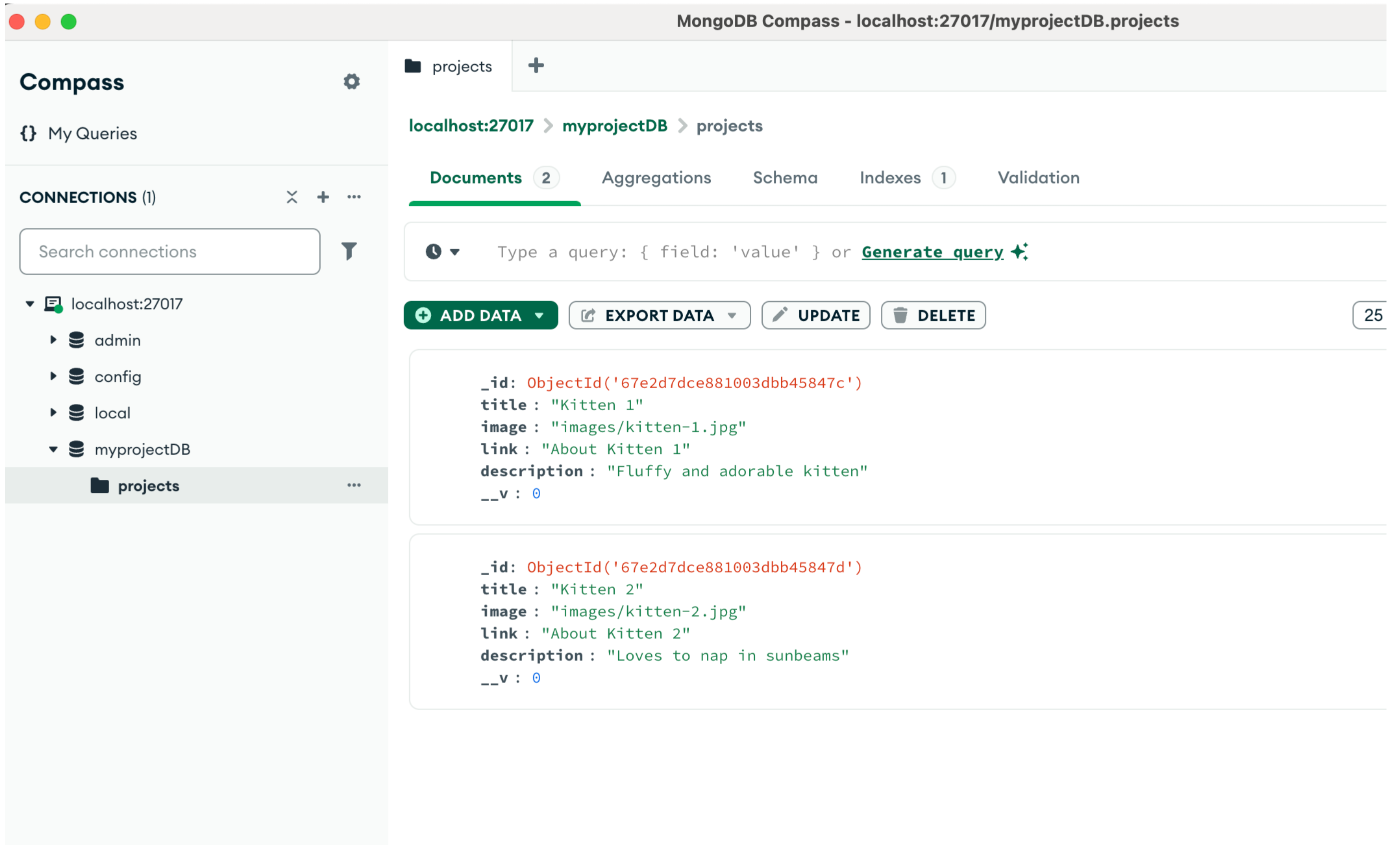
```
mongosh
```

Then use:

```
use myprojectDB
db.projects.find().pretty()
```

Option 2: Use Compass (MongoDB GUI)

Download and open MongoDB Compass.
Connect to: mongodb://localhost:27017
Find the database myprojectDB
Click on the projects collection and view your documents.



```
_id: ObjectId('67e2d7dce881003dbb45847c')
title : "Kitten 1"
image : "images/kitten-1.jpg"
link : "About Kitten 1"
description : "Fluffy and adorable kitten"
__v : 0
```

```
_id: ObjectId('67e2d7dce881003dbb45847d')
title : "Kitten 2"
image : "images/kitten-2.jpg"
link : "About Kitten 2"
description : "Loves to nap in sunbeams"
__v : 0
```

Thanks