Contemporary Issues in Software Engineering
Semester 2, 2024

Worksheet 3 Ass1A: The MNNN Stack

Worksheet Instructions

Deliverables and Due dates:

You are required to complete the Worksheet and keep evidence as you do it by selectively taking screenshots of your work, as well as explanations.

Each worksheet should ideally be checked off by the TA by the end of the week's (week four} tutorial

This worksheet should be Checked off and Uploaded on CANVAS ideally by end of Tutorial Week 4 – all four worksheets for Assignment 1a are due by week 6, and the knowledge develops cumulatively so don't leave it to the end – that will also make it hard for the TA's to mark and give you feedback.

EXPERIMENT – BE CURIOUS – TEACH OTHERS – TAKE SELECTED SCREENSHOTS FOR KEY ASPECTS

The worksheets will have some theory, a practical exercise, and a worksheet for answers to questions and at least three selectively captured screenshots as evidence. The aim is to be able to learn from the exercise, and evidence that.

For each of your **three selected screenshots** (or sequence of shots) in a brief paragraph or two reflect on why you have selected it. What have you learnt in this part of the worksheet? What was new or surprising? What useful external resource(s) did you consult and why? Provide a link(s) to the resource.

===

This worksheet will bring together a group of ideas to create a simple MNNN web app.

The worksheet relies on you working through the tutorial and reading some online resources to achieve the desired skills so we can make progress on the Software Practice Empirical Evidence Database (SPEED) app for Peter the PO.

What we need to be able to do for CISE and the SPEED product development

There is a lot that *could* be learned about using the MNNN stack to create dynamic web apps. In this course you will need to learn just enough to deliver *some* functionality for the Software Empirical Evidence Database (SPEED)app. This will involve some simple forms and data display as a minimum so that the Product Owner (PO) can test the idea out.

The SPEED application should allow practitioners to be able to find the level of evidence that there is in academic articles that supports or refutes the claimed benefits of different SE practices (eg. what is the level of evidence that using TDD will improve code quality -compared to not using TDD). The SPEED app needs the following functionality to test the idea out:

1. A way to select an SE practice (e.g., TDD) that we want to get the benefits that are claimed about, as well as the level of evidence (eg. strongly supports, weakly refutes) for those claimed benefits.
    a. The user should only be able to select from SE practices that we have evidence of claimed benefits for in our database.
    b. This could be a dropdown list of SE practices that are in our database and that we can select from to return the list of claims and evidence from our database for the selected SE Practice (from the dropdown).
        i. One way to do this is to use the "react-select" library in the React component library.

2. A way to view the level of evidence for each claimed benefit for the SE Practice selected.
    a. It will be useful for the user to see other information about the source of the evidence (the article analysed by the analyst and submitted by a submitter) -such as article title, article authors, journal name, year of publication.
    b. This could be a table of data that can be sorted and filtered.
        i. Each row would be evidence of different claimed benefit from a specific article, each column would be a different piece of info about that claimed benefit and evidence (title, etc)
        ii. The table should be sortable on any column
        iii. The table should be filterable by claimed benefit (and maybe year range of publication, and maybe level of evidence)
        iv. One way to create a table in React is using the react-table library
3. An input form for a submitter (anyone) to submit an article they think is relevant for our evidence database.
    a. The article should be about a SE practice and have some evidence about a claimed benefit(s) for this SE practice.
    b. This article may or may not be accepted (moderator will decide)
    c. If it is accepted then the analyst will read the article and decide on the level of evidence for each claimed benefit in the article and enter that into our SPEED database.

===

<u>**How to learn to do the things we need to do.**</u>

This course is about the practices around the coding (CI/CD, TDD, mob/pair programming, planning, retrospectives, code reviews, code quality testing etc), so we do **not** want to spend too much effort becoming expert in JavaScript/Typescript or React or Express/Nest/Node. Rather we just want to learn **enough** so we understand the **principles and can create basic functionality.**

In industry it is common to have to learn a new language or framework or tool – they are changing almost weekly, so this is a useful skill to develop – learning how to learn from diverse sources.

To learn quickly it is useful to watch others model the thinking and coding – **BUT only if you take notes.** Otherwise it is too passive, and you will not retain or learn much.

Then the ONLY way to make this useful so YOU can do it, is to **actually do it yourself !**

This is why we are using worksheets to encourage you to read, watch **and do**.

Many people in industry learn from tutorials online or take online courses, and I am sure many of you do this already. It is part of the new way of learning – not just classroom learning, but the sources of information and practice can also be from podcasts, blogs, videos, websites and so on.

The problem is there are worthless, out-of-date, or even plain wrong videos and tutorials on the web. And there are LOTS of videos and tutorials to sift through. The problem becomes finding the good ones. Hopefully you find the resources we direct you to useful.

The MNNN Tutorial

Contents

This tutorial is adopted from "The MERN stack tutorial" by Nur Islam on LogRocket. It is recommended to also read the original post for extra information and comparison between MERN and MNNN.
Do not simply copy and paste the code in this tutorial, you should read and understand each line, as you will need to build your own functionality(s) later. Without a clear understanding of the code, it may be hard.

What is the MNNN stack?

- MongoDB: A cross-platform document-oriented database program.
- Nest.js: A progressive Node.js framework for building efficient, reliable and scalable server-side applications.
- Next.js: A React framework for building full-stack web applications. (Usually we do not use it to build backend)
- Node.js: An open source, cross-platform, JavaScript runtime environment that executes JavaScript code outside of a browser.

MongoDB, Nest.js and Node.js help you build the backend, and Next.js powers the frontend. Make sure you have Node.js installed before we start.

Server setup with Nest.js and Node.js

Backend app initialization

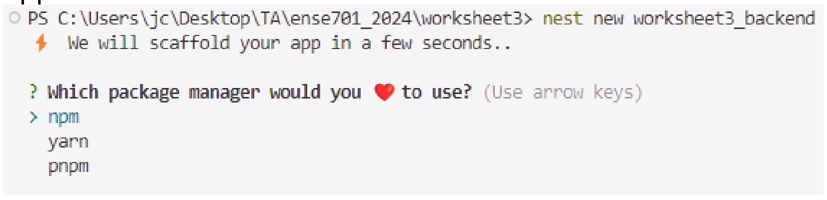
First, install the Nest.js if you haven't, by running the terminal command:

```
$ npm i -g @nestjs/cli
```

Now we initialize the backend app. In our working folder (`worksheet3/`) run terminal command:

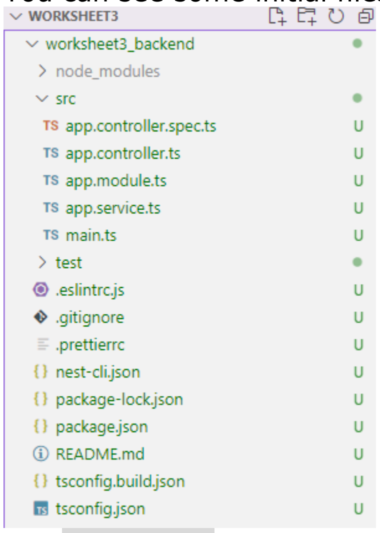
```
$ nest new project-name
```

I call it worksheet3_backend, so the command is:
$ nest new worksheet3_backend
Select whatever package manager you want to use, or just use npm by default. Then the initialization script will start to download a set of default packages to support the basic Nest.js application.


```
○ PS C:\Users\jc\Desktop\TA\ense701_2024\worksheet3> nest new worksheet3_backend
  ⚡  We will scaffold your app in a few seconds..

  ? Which package manager would you ❤️ to use? (Use arrow keys)
  > npm
    yarn
    pnpm
```

Setting the entry point

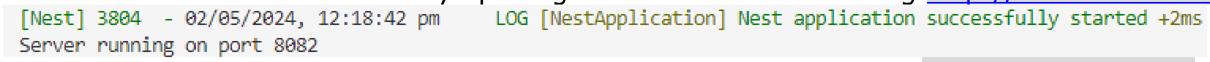
Once the initialization has finished, go into the backend folder (worksheet3_backend/ in my case). You can see some initial files:


```
WORKSHEET3
  worksheet3_backend
    > node_modules
    src
      TS app.controller.spec.ts   U
      TS app.controller.ts        U
      TS app.module.ts            U
      TS app.service.ts           U
      TS main.ts                  U
    > test
    .eslintrc.js                  U
    .gitignore                    U
    .prettierrc                   U
    nest-cli.json                 U
    package-lock.json             U
    package.json                  U
    README.md                     U
    tsconfig.build.json           U
    tsconfig.json                 U
```

The src/main.ts is the entry of our backend application. You can find more information about these files and their roles in Nest's official documents.
Let's configure the port of the app. Open main.ts, change the code to these:

```
import { NestFactory } from '@nestjs/core';
import { AppModule } from './app.module';

async function bootstrap() {
  const app = await NestFactory.create(AppModule);
  const port = process.env.PORT || 8082;
  await app.listen(port, () => console.log(`Server running on port ${port}`));
}
bootstrap();
```

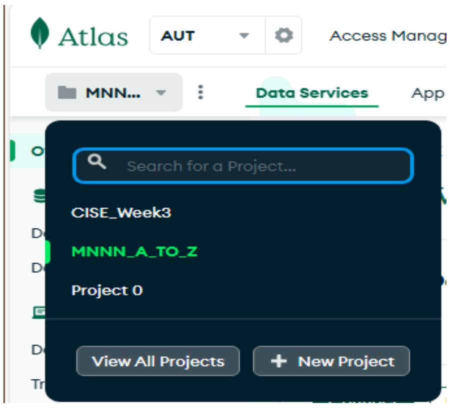
After that, run the $ npm run start command. You will see Server running on port 8082. You can also check it from the browser by opening the browser and entering http://localhost:8082.


```
[Nest] 3804  - 02/05/2024, 12:18:42 pm     LOG [NestApplication] Nest application successfully started +2ms
Server running on port 8082
```

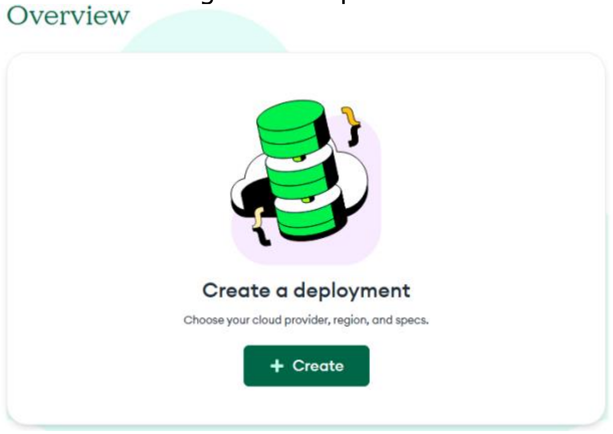
You can also run the backend app in development mode by running $ npm run start:dev, so that the changes to the source files will trigger a restart of the app automatically.

Database setup with MongoDB

MongoDB provides a multi-cloud database service known as Atlas, which simplifies the process of deploying and managing MongoDB databases. We will use Atlas to create the database for our MERN application.
To begin, log on to your MongoDB Atlas dashboard and initiate a new project. Feel free to assign it any name; in our case, we'll name it MNNN_A_TO_Z.



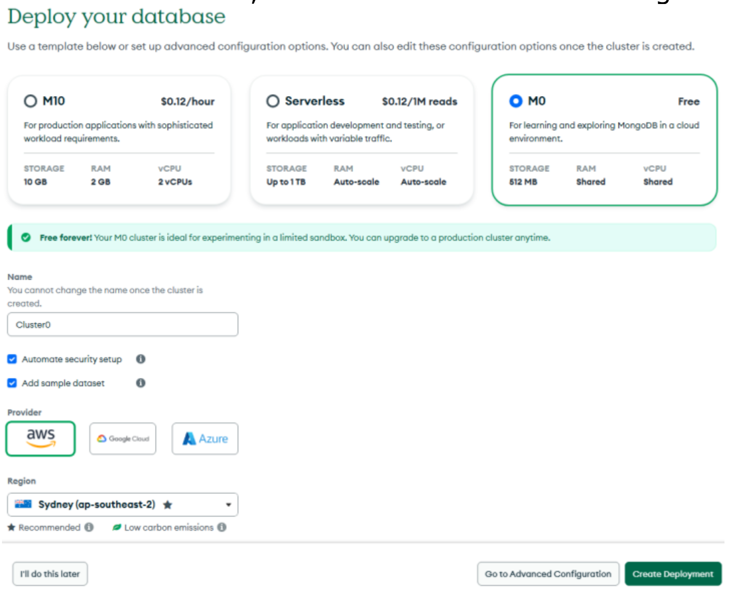
Next, create a new database cluster by clicking the create button and selecting the desired plan. We will be using the free plan.


Overview

Create a deployment

Choose your cloud provider, region, and specs.

On the same page, you have the option of selecting a preferred provider, region, and name for the database. For now, we will use the default settings:


Deploy your database

Use a template below or set up advanced configuration options. You can also edit these configuration options once the cluster is created.

○ M10	$0.12/hour	For production applications with sophisticated workload requirements.	STORAGE 10 GB	RAM 2 GB
○ Serverless	$0.12/1M reads	For application development and testing, or workloads with variable traffic.	STORAGE Up to 1 TB	RAM Auto-scale
● M0	Free	For learning and exploring MongoDB in a cloud environment.	STORAGE 512 MB	RAM Shared

vCPU: M10 2 vCPUs; Serverless Auto-scale; M0 Shared

Free forever! Your M0 cluster is ideal for experimenting in a limited sandbox. You can upgrade to a production cluster anytime.

Name
You cannot change the name once the cluster is created.
Cluster0

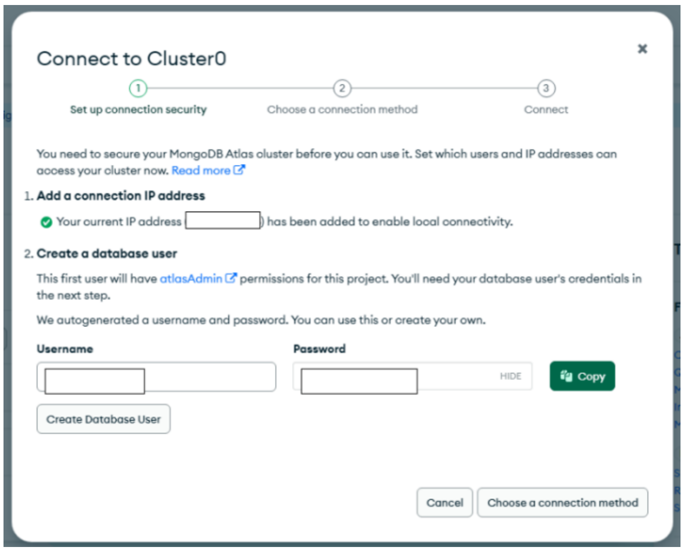
☑ Automate security setup
☑ Add sample dataset

Provider
[✓ aws] Google Cloud Azure

Region
Sydney (ap-southeast-2) ★

★ Recommended   Low carbon emissions

After clicking the Create button, the creation process will begin in the background.
While the cluster is being created, you will be presented with a connection quickstart. This quickstart allows you to create a user and enables your IP address to access your cluster:



Connect to Cluster0

You need to secure your MongoDB Atlas cluster before you can use it. Set which users and IP addresses can access your cluster now. Read more

1. **Add a connection IP address**

Your current IP address [redacted] has been added to enable local connectivity.

2. **Create a database user**

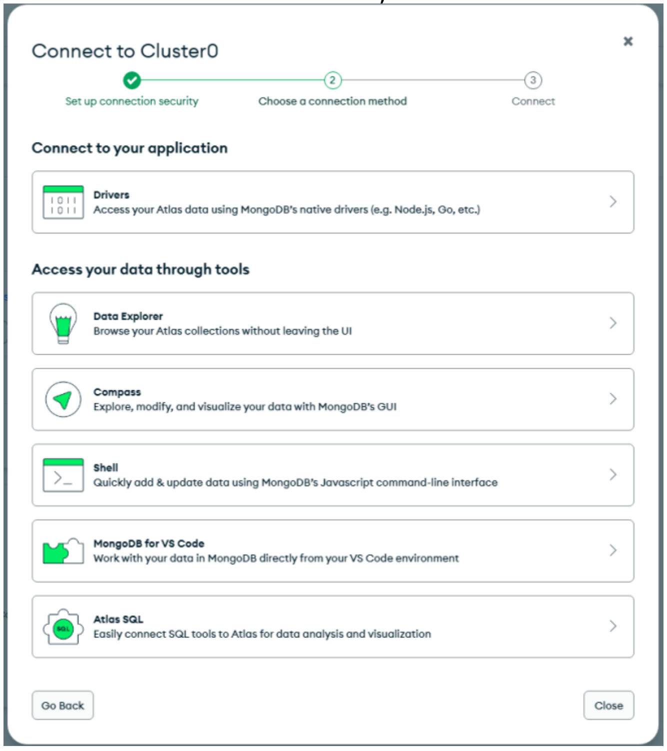
This first user will have atlasAdmin permissions for this project. You'll need your database user's credentials in the next step.

We autogenerated a username and password. You can use this or create your own.

Username [redacted]

Password [redacted]

First create a database user, then choose a connection method:



Connect to Cluster0

Connect to your application

Drivers
Access your Atlas data using MongoDB's native drivers (e.g. Node.js, Go, etc.)

Access your data through tools

Data Explorer
Browse your Atlas collections without leaving the UI

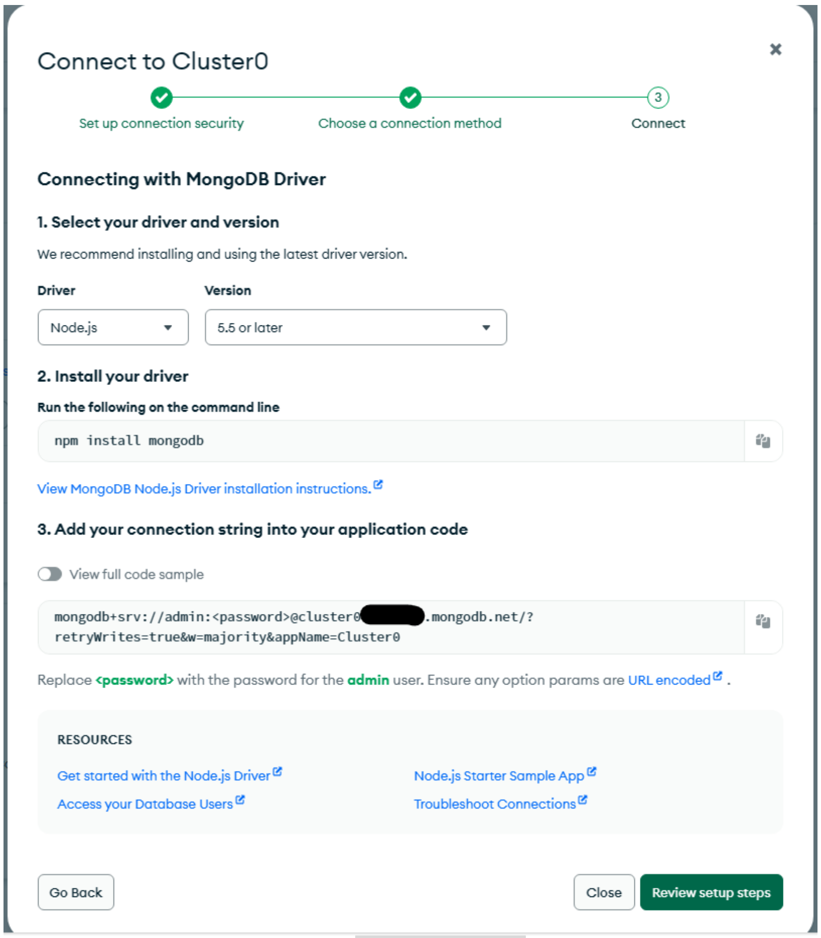
Compass
Explore, modify, and visualize your data with MongoDB's GUI

Shell
Quickly add & update data using MongoDB's Javascript command-line interface

MongoDB for VS Code
Work with your data in MongoDB directly from your VS Code environment

Atlas SQL
Easily connect SQL tools to Atlas for data analysis and visualization

Select the Drivers, then you will see the connection string, which is needed to connect to our cluster (database):



Connecting with MongoDB Driver

1. Select your driver and version

We recommend installing and using the latest driver version.

Driver: Node.js
Version: 5.5 or later

2. Install your driver

Run the following on the command line

```
npm install mongodb
```

View MongoDB Node.js Driver installation instructions.

3. Add your connection string into your application code

View full code sample

```
mongodb+srv://admin:<password>@cluster0[redacted].mongodb.net/?
retryWrites=true&w=majority&appName=Cluster0
```

Replace **<password>** with the password for the **admin** user. Ensure any option params are URL encoded .

RESOURCES

Get started with the Node.js Driver
Node.js Starter Sample App
Access your Database Users
Troubleshoot Connections

Remember to replace the <password> with the user's password. Now the database is set and running, and we are ready to connect to it.
Although not required for this tutorial, as we are working on a small project, it is important to understand MongoDB's advanced querying techniques, such as aggregation pipelines. These pipelines allow us to perform complex manipulations and transformations on data sets directly in the database, providing powerful querying capabilities, including document grouping, filtering, transformation, and the calculation of aggregate values.
Additionally, optimizing the database performance is essential, especially when scaling your application and for large-scale MERN applications, as this improves the performance of your application. MongoDB provides various methods to achieve this, including but not limited to:

- Indexes: Implementing appropriate indexes on frequently queried fields can significantly enhance query performance
- Sharding: Sharding distributes data across multiple servers, thereby improving scalability
- Caching strategies: Utilizing caching mechanisms like Redis or in-memory caching can reduce database load and enhance overall application performance
- Query optimization: Crafting efficient queries and leveraging MongoDB's query planner to analyze and improve query execution plans
- Document structure: Designing an optimal document structure by embedding related data and avoiding complex joins can improve performance

Adding the database to our project

Now we can connect the database to our project.
We will need to install extra packages for database access:

```
$ npm i @nestjs/mongoose mongoose
```

Nest provides a simplified method to connect to MongoDB by using its mongoose module, you can read more in here. Edit your src/app.module.ts:

```
import { Module } from '@nestjs/common';
import { AppController } from './app.controller';
import { AppService } from './app.service';
import { MongooseModule } from '@nestjs/mongoose';
```

```
// remember to replace the <username> and <password> with your credentials
const DB_URI =
  'mongodb+srv://<username>:<password>@cluster0.ktxx5bo.mongodb.net/?retryWrites=true&w
=majority&appName=Cluster0';

@Module({
  imports: [MongooseModule.forRoot(DB_URI)],
  controllers: [AppController],
  providers: [AppService],
})
export class AppModule {}
```

It is usually not recommended to hard code your database connection string. Using the environment variable (or configuration variable) is a better idea. First, we need to install the required package:

```
$ npm i --save @nestjs/config
```

Create a .env file under your backend folder, edit it:

```
DB_URI='mongodb+srv://<username>:<password>@cluster0.ktxx5bo.mongodb.net/?retryWrites=t
rue&w=majority&appName=Cluster0'
```

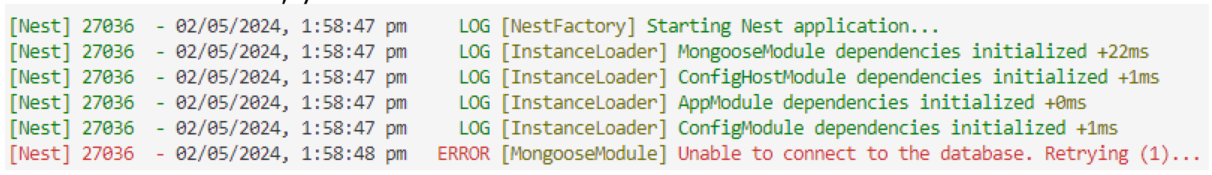
Again, remember to replace <username> and <password> with your credentials. If you want to use a specific database, you can specify it in the connection string; otherwise, the default 'test' database will be used:

```
DB_URI='mongodb+srv://<username>:<password>@cluster0.ktxx5bo.mongodb.net/THE_DATABSE_YO
U_WANT_TO_USE?retryWrites=true&w=majority&appName=Cluster0'
```

Edit your code in src/app.module.ts to use the environment variable:

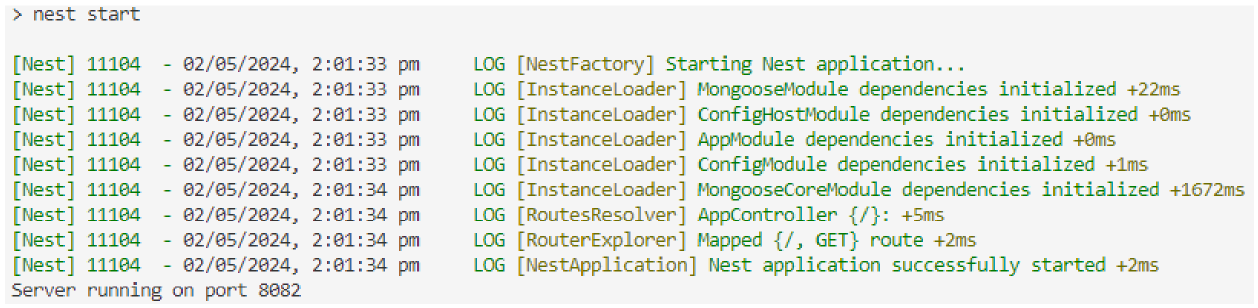
```
import { Module } from '@nestjs/common';
import { AppController } from './app.controller';
import { AppService } from './app.service';
import { MongooseModule } from '@nestjs/mongoose';
import { ConfigModule } from '@nestjs/config';

@Module({
  imports: [
    ConfigModule.forRoot(), MongooseModule.forRoot(process.env.DB_URI)],
  controllers: [AppController],
  providers: [AppService],
})
export class AppModule {}
```

Now you can run the app using the $ npm run start command. If there is anything wrong with your database connection, you will see:



```
[Nest] 27036  - 02/05/2024, 1:58:47 pm     LOG [NestFactory] Starting Nest application...
[Nest] 27036  - 02/05/2024, 1:58:47 pm     LOG [InstanceLoader] MongooseModule dependencies initialized +22ms
[Nest] 27036  - 02/05/2024, 1:58:47 pm     LOG [InstanceLoader] ConfigHostModule dependencies initialized +1ms
[Nest] 27036  - 02/05/2024, 1:58:47 pm     LOG [InstanceLoader] AppModule dependencies initialized +0ms
[Nest] 27036  - 02/05/2024, 1:58:47 pm     LOG [InstanceLoader] ConfigModule dependencies initialized +1ms
[Nest] 27036  - 02/05/2024, 1:58:48 pm   ERROR [MongooseModule] Unable to connect to the database. Retrying (1)...
```

Otherwise, it should run without issue:



```
> nest start

[Nest] 11104  - 02/05/2024, 2:01:33 pm     LOG [NestFactory] Starting Nest application...
[Nest] 11104  - 02/05/2024, 2:01:33 pm     LOG [InstanceLoader] MongooseModule dependencies initialized +22ms
[Nest] 11104  - 02/05/2024, 2:01:33 pm     LOG [InstanceLoader] ConfigHostModule dependencies initialized +0ms
[Nest] 11104  - 02/05/2024, 2:01:33 pm     LOG [InstanceLoader] AppModule dependencies initialized +0ms
[Nest] 11104  - 02/05/2024, 2:01:33 pm     LOG [InstanceLoader] ConfigModule dependencies initialized +1ms
[Nest] 11104  - 02/05/2024, 2:01:34 pm     LOG [InstanceLoader] MongooseCoreModule dependencies initialized +1672ms
[Nest] 11104  - 02/05/2024, 2:01:34 pm     LOG [RoutesResolver] AppController {/}: +5ms
[Nest] 11104  - 02/05/2024, 2:01:34 pm     LOG [RouterExplorer] Mapped {/, GET} route +2ms
[Nest] 11104  - 02/05/2024, 2:01:34 pm     LOG [NestApplication] Nest application successfully started +2ms
Server running on port 8082
```

Great! So far, we are on the right track, and our database is successfully connected. Now, time to complete the route setup, and after that, we will see how to create RESTful APIs.

Building RESTful APIs with the MNNN stack

To get started, create a folder named api under your backend src folder. It will hold all our APIs.
Inside the api folder, create a books folder to hold our books APIs.
First, we create database schemas. Create a file named book.schema.ts, and edit it:

```
import { Prop, Schema, SchemaFactory } from '@nestjs/mongoose';
import { Date, HydratedDocument } from 'mongoose';

export type BookDocument = HydratedDocument<Book>;

@Schema()
export class Book {
  @Prop({ required: true })
  title: string;

  @Prop({ required: true })
  isbn: string;

  @Prop({ required: true })
  author: string;

  @Prop()
  description: string;

  @Prop({ type: Date })
  published_date: Date;

  @Prop()
  publisher: string;

  @Prop({ type: Date, default: Date.now })
  updated_date: Date;
}

export const BookSchema = SchemaFactory.createForClass(Book);
```

Then we create a DTO file named create-book.dto.ts to handle the data transfer between frontend and backend:

```
import { Date } from 'mongoose';

export class CreateBookDto {
  title: string;
  isbn: string;
  author: string;
  description: string;
  published_date: Date;
  publisher: string;
  updated_date: Date;
}
```

Next, we will create the service (provider), controller, and module files for books API. The controller handles requests, the provider provide actual service (handles business logic), while the module packs the controller and service as a feature module. It is recommended to read more about them in the official document via the link.
book.service.ts:

```
import { Injectable } from '@nestjs/common';
import { Book } from './book.schema';
import { InjectModel } from '@nestjs/mongoose';
import { Model } from 'mongoose';
import { CreateBookDto } from './create-book.dto';

@Injectable()
export class BookService {
  constructor(@InjectModel(Book.name) private bookModel: Model<Book>) {}

  test(): string {
    return 'book route testing';
  }

  async findAll(): Promise<Book[]> {
    return await this.bookModel.find().exec();
  }

  async findOne(id: string): Promise<Book> {
    return await this.bookModel.findById(id).exec();
  }

  async create(createBookDto: CreateBookDto) {
    return await this.bookModel.create(createBookDto);
  }

  async update(id: string, createBookDto: CreateBookDto) {
    return await this.bookModel.findByIdAndUpdate(id, createBookDto).exec();
  }

  async delete(id: string) {
    const deletedBook = await this.bookModel.findByIdAndDelete(id).exec();
    return deletedBook;
  }
}
```

book.controller.ts:

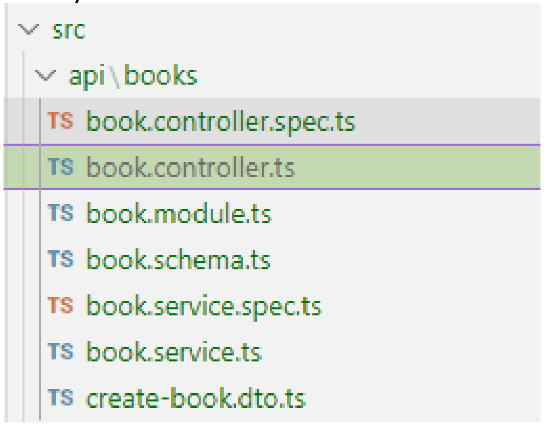
```
import {
  Body,
  Controller,
  Delete,
  Get,
  HttpException,
  HttpStatus,
  Param,
  Post,
  Put,
} from '@nestjs/common';
import { BookService } from './book.service';
import { CreateBookDto } from './create-book.dto';
import { error } from 'console';

@Controller('api/books')
export class BookController {
  constructor(private readonly bookService: BookService) {}

  @Get('/test')
  test() {
    return this.bookService.test();
  }
  // Get all books
  @Get('/')
  async findAll() {
    try {
      return this.bookService.findAll();
    } catch {
      throw new HttpException(
        {
          status: HttpStatus.NOT_FOUND,
          error: 'No Books found',
        },
        HttpStatus.NOT_FOUND,
        { cause: error },
      );
    }
  }

  // Get one book via id
  @Get('/:id')
  async findOne(@Param('id') id: string) {
    try {
      return this.bookService.findOne(id);
    } catch {
      throw new HttpException(
        {
          status: HttpStatus.NOT_FOUND,
          error: 'No Book found',
        },
        HttpStatus.NOT_FOUND,
        { cause: error },
      );
    }
  }
```

```
// Create/add a book
@Post('/')
async addBook(@Body() createBookDto: CreateBookDto) {
  try {
    await this.bookService.create(createBookDto);
    return { message: 'Book added successfully' };
  } catch {
    throw new HttpException(
      {
        status: HttpStatus.BAD_REQUEST,
        error: 'Unable to add this book',
      },
      HttpStatus.BAD_REQUEST,
      { cause: error },
    );
  }
}

// Update a book
@Put('/:id')
async updateBook(
  @Param('id') id: string,
  @Body() createBookDto: CreateBookDto,
) {
  try {
    await this.bookService.update(id, createBookDto);
    return { message: 'Book updated successfully' };
  } catch {
    throw new HttpException(
      {
        status: HttpStatus.BAD_REQUEST,
        error: 'Unable to update this book',
      },
      HttpStatus.BAD_REQUEST,
      { cause: error },
    );
  }
}

// Delete a book via id
@Delete('/:id')
async deleteBook(@Param('id') id: string) {
  try {
    return await await this.bookService.delete(id);
  } catch {
    throw new HttpException(
      {
        status: HttpStatus.NOT_FOUND,
        error: 'No such a book',
      },
      HttpStatus.NOT_FOUND,
      { cause: error },
      );
    }
  }
}
```

book.module.ts:

```
import { Module } from '@nestjs/common';
import { BookController } from './book.controller';
import { BookService } from './book.service';
import { MongooseModule } from '@nestjs/mongoose';
import { Book, BookSchema } from './book.schema';

@Module({
  imports: [
    MongooseModule.forFeature([{ name: Book.name, schema: BookSchema }]),
  ],
  controllers: [BookController],
  providers: [BookService],
})
export class BookModule {}
```

Now your books folder should look like this:



```
src
  api\books
    book.controller.spec.ts
    book.controller.ts
    book.module.ts
    book.schema.ts
    book.service.spec.ts
    book.service.ts
    create-book.dto.ts
```

Then we need to import the books module into the app module (src/app.module.ts), so that the backend app knows it should load this book module (i.e. our books API) when it starts:

```
import { Module } from '@nestjs/common';
import { AppController } from './app.controller';
import { AppService } from './app.service';
import { MongooseModule } from '@nestjs/mongoose';
import { BookModule } from './api/books/book.module';
import { ConfigModule } from '@nestjs/config';

@Module({
  imports: [
    ConfigModule.forRoot(),
    MongooseModule.forRoot(process.env.DB_URI),
    BookModule,
  ],
  controllers: [AppController],
  providers: [AppService],
})
export class AppModule {}
```

Now the backend is all set, except the CORS. Here is a good explanation of it. We will deal with it later when the frontend is ready.
When the backend app is running, you can use tools such as Postman to simulate some requests, and see if you can interact with the database by adding or getting books.

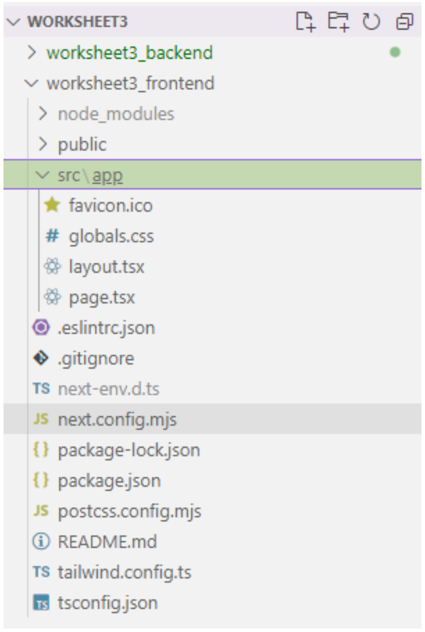
Building the frontend

Setting up a Next.js app

Let’s create a new Next.js app by running:
`$ npx create-next-app@latest project-name`
I call the project `worksheet3_frontend`.
You will be asked to choose from a set of options, usually it is ok to use the default setting:

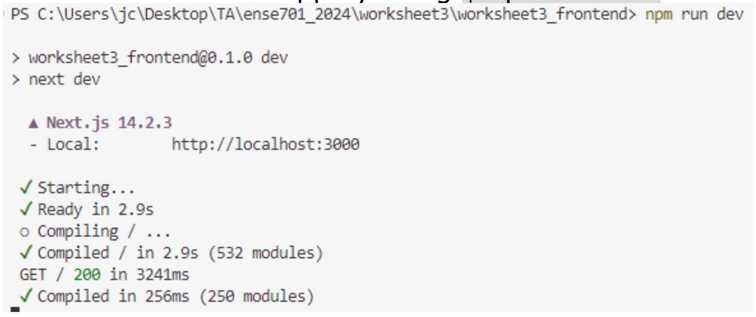
```
PS C:\Users\jc\Desktop\TA\ense701_2024\worksheet3> npx create-next-app@latest worksheet3_f
rontend
√ Would you like to use TypeScript? ... No / [✓ Yes]
√ Would you like to use ESLint? ... No / [✓ Yes]
√ Would you like to use Tailwind CSS? ... No / [✓ Yes]
√ Would you like to use `src/` directory? ... No / [✓ Yes]
√ Would you like to use App Router? (recommended) ... No / [✓ Yes]
√ Would you like to customize the default import alias (@/*)? ... [✓ No] / Yes
Creating a new Next.js app in C:\Users\jc\Desktop\TA\ense701_2024\worksheet3\worksheet3_fr
ontend.
```

The script will automatically install a set of packages. Once it is finished, your frontend folder should look like this:



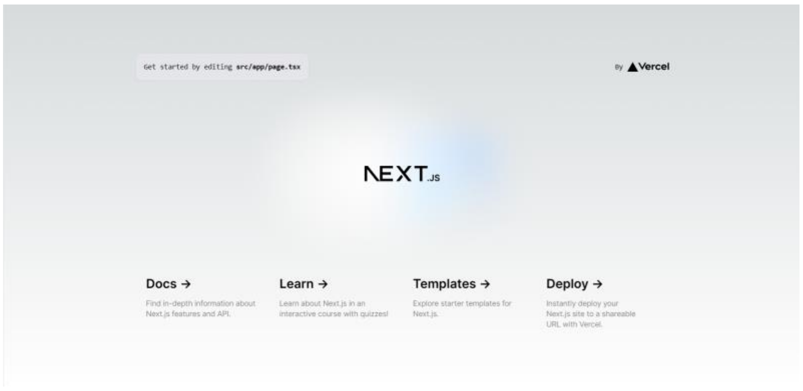
```
WORKSHEET3
  > worksheet3_backend
  v worksheet3_frontend
    > node_modules
    > public
    v src\app
      favicon.ico
      globals.css
      layout.tsx
      page.tsx
    .eslintrc.json
    .gitignore
    next-env.d.ts
    next.config.mjs
    package-lock.json
    package.json
    postcss.config.mjs
    README.md
    tailwind.config.ts
    tsconfig.json
```

You can read more about the initial files and their roles in here. We are using the app routing here, which we will talk about later.
Now we can run the app by using `$ npm run dev` command:



```
PS C:\Users\jc\Desktop\TA\ense701_2024\worksheet3\worksheet3_frontend> npm run dev

> worksheet3_frontend@0.1.0 dev
> next dev

  ▲ Next.js 14.2.3
  - Local:        http://localhost:3000

 ✓ Starting...
 ✓ Ready in 2.9s
 ○ Compiling / ...
 ✓ Compiled / in 2.9s (532 modules)
 GET / 200 in 3241ms
 ✓ Compiled in 256ms (250 modules)
```

And we can access the default frontend app in your browser at `localhost:3000`:


Get started by editing src/app/page.tsx

Find in-depth information about Next.js features and API.

Learn about Next.js in an interactive course with quizzes!

Explore starter templates for Next.js.

Instantly deploy your Next.js site to a shareable URL with Vercel.

Adding Bootstrap to your React app

We have our initial setup file for the frontend part. Now, we can start integrating our backend with our frontend. Before that, I want to add Bootstrap to our project.
First, you'll need to install Bootstrap as a dependency of the project using the following command:

```
$ npm i bootstrap
```

It is a bit tricky to use bootstrap in Next.js due to its server-side rendering (SSR). But there are workarounds. This one is what I prefer.
First, we create a folder called `components` under the `src` folder to contain all components we want to build. Then, under `src/components/` create a `BootstrapClient.ts` and add codes:

```
"use client"

import { useEffect } from "react";
function BootstrapClient() {
    useEffect(() => {
        require("bootstrap/dist/js/bootstrap.bundle.min.js");
    }, []);

    return null;
}

export default BootstrapClient;
```

Then edit the `src/app/layout.tsx`:

```
import type { Metadata } from "next";
import { Inter } from "next/font/google";
import "bootstrap/dist/css/bootstrap.css";
import "./globals.css";
import BootstrapClient from "@/components/BootstrapClient";

const inter = Inter({ subsets: ["latin"] });

export const metadata: Metadata = {
  title: "Create Next App",
  description: "Generated by create next app",
};

export default function RootLayout({
  children,
}: Readonly<{
  children: React.ReactNode;
}>) {
  return (
    <html lang="en">
```

```
      <body className={inter.className}>
        <BootstrapClient />
        {children}
      </body>
    </html>
  );
}
```

Now Bootstrap is enabled everywhere.

Creating the components

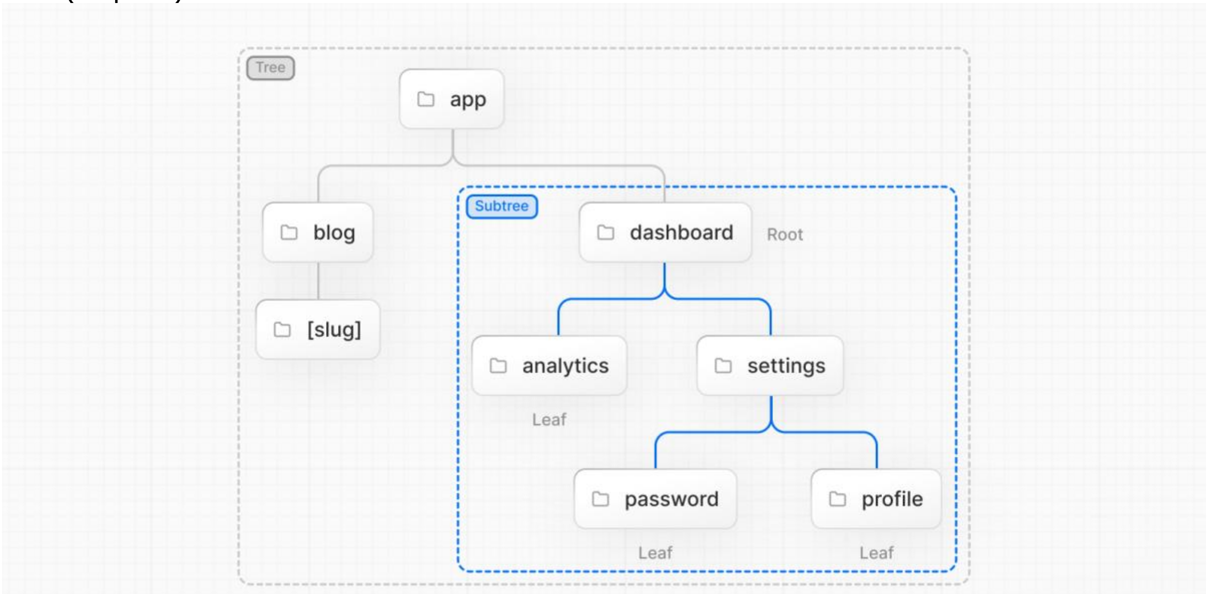
Inside the src/components folder create five different files:

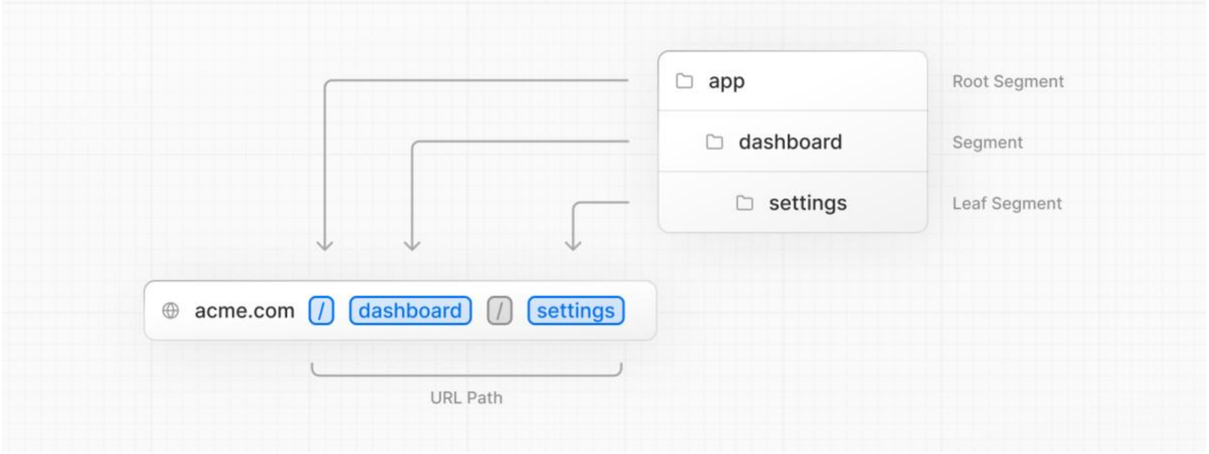
1. CreateBook.tsx
2. ShowBookList.tsx
3. BookCard.tsx
4. ShowBookDetails.tsx
5. UpdateBookInfo.tsx

We will work with these five files a bit later.

Setting up routes

It is strongly recommended to read the routing fundamentals in Next.js.
The images below show the structure of folders. At the same time, it is also the structure of the URL (or path).





* Both images are referenced from Building Your Application: Routing | Next.js (nextjs.org)

Under each folder (start from app folder), there should be a page.tsx file that contains the actual component of that page. We can think of it in this way, when we try to access localhost:3000/, the app will try to find the app/page.tsx, and display the content to you; if we try to access localhost:3000/mypage, then the app will try to find the app/mypage/page.tsx. Note that this is NOT what Next.js actually does when you access the page; it actually packs everything together when building (compiling) the app for a faster access.
Back to our app. In the main page (homepage) we display all the books. Edit the app/page.tsx:

```
'use client'

import ShowBookList from "@/components/ShowBookList";

export default function Home() {
  return (
    <main>
      <ShowBookList />
    </main>
  );
}
```

Then, we create three folders under the app folder:

1. create-book
2. edit-book
3. show-book

The create-book is used for creating a new book. The edit-book is for editing an existing book (via id). The show-book is used to show a specific book (via id).
Under the app/create-book folder, create a page.tsx:

```
'use client'

import CreateBookComponent from "@/components/CreateBook";

export default function CreateBook() {
  return (
    <main>
      <CreateBookComponent />
    </main>
  );
}
```

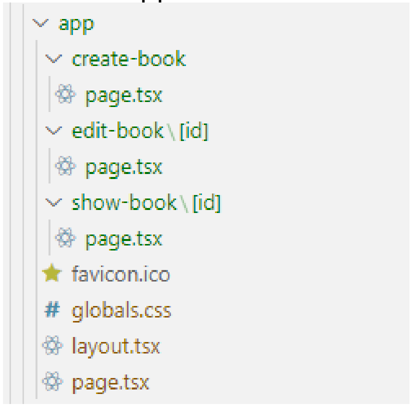
The handling of edit-book and show-book is a bit different. There is a dynamic element in the path (URL), the 'id'. For example, if we want to edit a book for which the id is '15', we will access the URL: localhost:3000/edit-book/15. The id can change, so how can Next.js handle it by using the folder structure? There is a mechanism called dynamic routes.
We create a folder called [id] under the app/edit-book and app/show-book respectively. In this way, Next.js knows that this [id] is dynamic. And we can access the value of this [id] by using the useParams() function. Then we create page.tsx under both [id] folders.
app/edit-book/[id]/page.tsx:

```
'use client'

import UpdateBookInfo from "@/components/UpdateBookInfo";

export default function ShowBook() {
  return (
    <UpdateBookInfo />
  )
}
```

app/show-book/[id]/page.tsx:

```
'use client'

import ShowBookDetails from "@/components/ShowBookDetails";

export default function ShowBook() {
  return (
    <ShowBookDetails />
  )
}
```

Now our app folder should look like this:



```
app
  create-book
    page.tsx
  edit-book\[id]
    page.tsx
  show-book\[id]
    page.tsx
  favicon.ico
  globals.css
  layout.tsx
  page.tsx
```

Adding our feature components

Now, it’s time to add feature components to our project. In the following codes, fetch() function and page navigation are used, it is recommended to read the document via the link. You may also consider to use async and await together with fetch() to provide a smoother user experience.

Book.ts

We first add this file to define a Book type. It will be used in all other components.

```
export type Book = {
    _id?: string;
    title?: string;
    isbn?: string;
    author?: string;
    description?: string;
    published_date?: Date;
    publisher?: string;
    updated_date?: Date;
};

export const DefaultEmptyBook: Book = {
    _id: undefined,
    title: '',
    isbn: '',
    author: '',
    description: '',
    published_date: undefined,
    publisher: '',
    updated_date: undefined,
}
```

CreateBook.tsx

Our CreateBook.tsx file (under the src/components folder) is responsible for adding, creating, or saving a new book or a book's info.

```
import React, { ChangeEvent, FormEvent, useState } from "react";
import { useRouter } from "next/navigation";
import Link from "next/link";
import { Book, DefaultEmptyBook } from "./Book";

const CreateBookComponent = () => {
  const navigate = useRouter();

  const [book, setBook] = useState<Book>(DefaultEmptyBook);

  const onChange = (event: ChangeEvent<HTMLInputElement>) => {
    setBook({ ...book, [event.target.name]: event.target.value });
  };

  const onSubmit = (event: FormEvent<HTMLFormElement>) => {
    event.preventDefault()
    console.log(book);
    fetch("http://localhost:8082/api/books", {method: 'POST', headers: {"Content-Type":
"application/json"}, body: JSON.stringify(book)})
      .then((res) => {
        console.log(res);
        setBook(DefaultEmptyBook);
        // Push to /
        navigate.push("/");
      })
      .catch((err) => {
        console.log('Error from CreateBook: ' + err);
      });
  };

  return (
    <div className="CreateBook">
      <div className="container">
        <div className="row">
          <div className="col-md-8 m-auto">
            <br />
            <Link href="/" className="btn btn-outline-warning float-left">
              Show BooK List
            </Link>
          </div>
          <div className="col-md-10 m-auto">
            <h1 className="display-4 text-center">Add Book</h1>
            <p className="lead text-center">Create new book</p>
            <form noValidate onSubmit={onSubmit}>
              <div className="form-group">
                <input
                  type="text"
                  placeholder="Title of the Book"
                  name="title"
                  className="form-control"
                  value={book.title}
                  onChange={onChange}
                />
              </div>
              <br />
              <div className="form-group">
                <input
                  type="text"
                  placeholder="ISBN"
                  name="isbn"
                  className="form-control"
                  value={book.isbn}
                  onChange={onChange}
                />
              </div>
              <br />
              <div className="form-group">
                <input
                  type="text"
                  placeholder="Author"
                  name="author"
                  className="form-control"
                  value={book.author}
                  onChange={onChange}
                />
              </div>
              <br />
              <div className="form-group">
                <input
                  type="text"
                  placeholder="Describe this book"
                  name="description"
                  className="form-control"
                  value={book.description}
                  onChange={onChange}
                />
              </div>
              <br />
              <div className="form-group">
                <input
                  type="date"
                  placeholder="published_date"
                  name="published_date"
                  className="form-control"
                  value={book.published_date?.toString()}
                  onChange={onChange}
                />
              </div>
              <br />
              <div className="form-group">
```

```
                <input
                  type="text"
                  placeholder="Publisher of this Book"
                  name="publisher"
                  className="form-control"
                  value={book.publisher}
                  onChange={onChange}
                />
              </div>
              <button
                type="submit"
                className="btn btn-outline-warning btn-block mt-4 mb-4 w-100"
              >
                Submit
              </button>
            </form>
          </div>
        </div>
      </div>
    </div>
  );
};

export default CreateBookComponent;
```

ShowBookList.tsx

The ShowBookList.tsx component will be responsible for showing all the books we already have stored in our database.

```
import React, { useState, useEffect } from 'react';
import Link from 'next/link';
import BookCard from './BookCard';
import { Book } from './Book';

function ShowBookList() {
  const [books, setBooks] = useState<[Book?]>([]);

  useEffect(() => {
    fetch('http://localhost:8082/api/books')
      .then((res) => {
        return res.json();
      })
      .then((books) => {
        setBooks(books);
      })
      .catch((err) => {
        console.log('Error from ShowBookList: ' + err);
      });
  }, []);

  const bookList =
    books.length === 0
      ? 'there is no book record!'
      : books.map((book, k) => <BookCard book={book} key={k} />);

  return (
    <div className='ShowBookList'>
      <div className='container'>
        <div className='row'>
          <div className='col-md-12'>
            <br />
            <h2 className='display-4 text-center'>Books List</h2>
          </div>

          <div className='col-md-11'>
            <Link
              href='/create-book'
              className='btn btn-outline-warning float-right'
            >
              + Add New Book
            </Link>
            <br />
            <br />
            <hr />
          </div>
        </div>

        <div className='list'>{bookList}</div>
      </div>
    </div>
  );
}

export default ShowBookList;
```

BookCard.tsx

Here, we use a functional component called BookCard.tsx, which takes a book's info from ShowBookList.tsx and makes a card for each book.

```
import React from 'react';
import { Book } from './Book';
import { useRouter } from 'next/navigation';

interface IProp {
  book?: Book;
}

const BookCard = ({ book }: IProp) => {
  const router = useRouter();
  if (book == undefined) {
    return null;
  }
  const onClick = () => {
    router.push(`/show-book/${book._id}`)
  };
  return (
    <div className='card-container' onClick={onClick}>
      <img
        src='https://images.unsplash.com/photo-1495446815901-a7297e633e8d'
        alt='Books'
        height={200}
      />
      <div className='desc'>
        <h2>
          {book.title}
        </h2>
        <h3>{book.author}</h3>
        <p>{book.description}</p>
      </div>
    </div>
  );
};

export default BookCard;
```

ShowBookDetails.tsx

The ShowBookDetails component has one task: it shows all the info we have about any book. We have both delete and edit buttons here to get access.

```
'use client'

import React, { useState, useEffect } from 'react';
import { useParams, useRouter } from 'next/navigation';
import { Book, DefaultEmptyBook } from './Book';
import Link from 'next/link';

function ShowBookDetails() {
  const [book, setBook] = useState<Book>(DefaultEmptyBook);

  const id = useParams<{ id: string }>().id;
  const navigate = useRouter();

  useEffect(() => {
    fetch(`http://localhost:8082/api/books/${id}`)
      .then((res) => {
        return res.json()
      })
      .then((json) => {
        setBook(json);
      })
      .catch((err) => {
        console.log('Error from ShowBookDetails: ' + err);
      });
  }, [id]);

  const onDeleteClick = (id: string) => {
    fetch(`http://localhost:8082/api/books/${id}`, { method: 'DELETE' })
      .then((res) => {
        navigate.push('/');
      })
      .catch((err) => {
        console.log('Error form ShowBookDetails_deleteClick: ' + err);
      });
  };

  const BookItem = (
    <div>
      <table className='table table-hover table-dark table-striped table-bordered'>
        <tbody>
          <tr>
            <th scope='row'>1</th>
            <td>Title</td>
            <td>{book.title}</td>
          </tr>
          <tr>
            <th scope='row'>2</th>
            <td>Author</td>
            <td>{book.author}</td>
          </tr>
          <tr>
            <th scope='row'>3</th>
            <td>ISBN</td>
            <td>{book.isbn}</td>
          </tr>
          <tr>
            <th scope='row'>4</th>
            <td>Publisher</td>
            <td>{book.publisher}</td>
          </tr>
          <tr>
            <th scope='row'>5</th>
            <td>Published Date</td>
            <td>{book.published_date?.toString()}</td>
          </tr>
          <tr>
            <th scope='row'>6</th>
            <td>Description</td>
            <td>{book.description}</td>
          </tr>
        </tbody>
      </table>
    </div>
  );

  return (
    <div className='ShowBookDetails'>
      <div className='container'>
        <div className='row'>
          <div className='col-md-10 m-auto'>
            <br /> <br />
            <Link href='/' className='btn btn-outline-warning float-left'>
              Show Book List
            </Link>
          </div>
          <br />
          <div className='col-md-8 m-auto'>
            <h1 className='display-4 text-center'>Book&quot;s Record</h1>
            <p className='lead text-center'>View Book&quot;s Info</p>
            <hr /> <br />
          </div>
          <div className='col-md-10 m-auto'>{BookItem}</div>
          <div className='col-md-6 m-auto'>
            <button
              type='button'
              className='btn btn-outline-danger btn-lg btn-block'
              onClick={() => {
                onDeleteClick(book._id || "");
              }}
            >
              Delete Book
            </button>
          </div>
          <div className='col-md-6 m-auto'>
            <Link
              href={`/edit-book/${book._id}`}
              className='btn btn-outline-info btn-lg btn-block'
            >
              Edit Book
            </Link>
          </div>
        </div>
      </div>
    </div>
  );
}

export default ShowBookDetails;
```

UpdateBookInfo.tsx

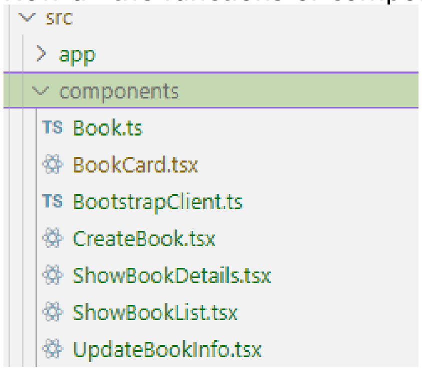
UpdateBookInfo.tsx, as its name indicates, is responsible for updating a book's info. An Edit Book button will trigger this component to perform. After clicking Edit Book, we will see a form with the old info, which we will be able to edit or replace.

```
import React, { useState, useEffect, ChangeEvent, FormEvent, ChangeEventHandler } from
'react';
import { useParams, useRouter } from 'next/navigation';
import { Book, DefaultEmptyBook } from './Book';
import Link from 'next/link';

function UpdateBookInfo() {
  const [book, setBook] = useState<Book>(DefaultEmptyBook);
  const id = useParams<{ id: string }>().id;
  const router = useRouter();

  useEffect(() => {
    fetch(`http://localhost:8082/api/books/${id}`)
```

```
      .then((res) => {
        return res.json();
      })
      .then((json) => {
        setBook(json);
      })
      .catch((err) => {
        console.log('Error from UpdateBookInfo: ' + err);
      });
  }, [id]);

  const inputOnChange = (event: ChangeEvent<HTMLInputElement>) => {
    setBook({ ...book, [event.target.name]: event.target.value });
  };

  const textAreaOnChange = (event: ChangeEvent<HTMLTextAreaElement>) => {
    setBook({ ...book, [event.target.name]: event.target.value });
  }

  const onSubmit = (event: FormEvent<HTMLFormElement>) => {
    event.preventDefault();

    fetch(`http://localhost:8082/api/books/${id}`, {method: 'PUT', headers: {"Content-
Type": "application/json"}, body: JSON.stringify(book)})
      .then((res) => {
        router.push(`/show-book/${id}`);
      })
      .catch((err) => {
        console.log('Error from UpdateBookInfo: ' + err);
      });
  };

  return (
    <div className='UpdateBookInfo'>
      <div className='container'>
        <div className='row'>
          <div className='col-md-8 m-auto'>
            <br />
            <Link href='/' className='btn btn-outline-warning float-left'>
              Show BooK List
            </Link>
          </div>
          <div className='col-md-8 m-auto'>
            <h1 className='display-4 text-center'>Edit Book</h1>
            <p className='lead text-center'>Update Book&quot;s Info</p>
          </div>
        </div>

        <div className='col-md-8 m-auto'>
          <form noValidate onSubmit={onSubmit}>
            <div className='form-group'>
              <label htmlFor='title'>Title</label>
              <input
```

```
                type='text'
                placeholder='Title of the Book'
                name='title'
                className='form-control'
                value={book.title}
                onChange={inputOnChange}
              />
            </div>
            <br />

            <div className='form-group'>
              <label htmlFor='isbn'>ISBN</label>
              <input
                type='text'
                placeholder='ISBN'
                name='isbn'
                className='form-control'
                value={book.isbn}
                onChange={inputOnChange}
              />
            </div>
            <br />

            <div className='form-group'>
              <label htmlFor='author'>Author</label>
              <input
                type='text'
                placeholder='Author'
                name='author'
                className='form-control'
                value={book.author}
                onChange={inputOnChange}
              />
            </div>
            <br />

            <div className='form-group'>
              <label htmlFor='description'>Description</label>
              <textarea
                placeholder='Description of the Book'
                name='description'
                className='form-control'
                value={book.description}
                onChange={textAreaOnChange}
              />
            </div>
            <br />

            <div className='form-group'>
              <label htmlFor='published_date'>Published Date</label>
              <input
                type='text'
                placeholder='Published Date'
                name='published_date'
                className='form-control'
                value={book.published_date?.toString()}
                onChange={inputOnChange}
              />
            </div>
            <br />

            <div className='form-group'>
              <label htmlFor='publisher'>Publisher</label>
              <input
                type='text'
                placeholder='Publisher of the Book'
                name='publisher'
                className='form-control'
                value={book.publisher}
                onChange={inputOnChange}
              />
            </div>
            <br />

            <button
              type='submit'
              className='btn btn-outline-info btn-lg btn-block'
            >
              Update Book
            </button>
          </form>
        </div>
      </div>
    </div>
  );
}

export default UpdateBookInfo;
```

Now all the functions of components are finished. Your src/components folder should look like this:



```
src
  app
  components
    Book.ts
    BookCard.tsx
    BootstrapClient.ts
    CreateBook.tsx
    ShowBookDetails.tsx
    ShowBookList.tsx
    UpdateBookInfo.tsx
```

Use ENV file

Since all the backend URLs are hardcoded in the fetch function, it can be annoying to modify them when the backend moves to a new URL. So let's use the ENV file. Create a .env under the frontend root folder, put this line of code:

```
NEXT_PUBLIC_BACKEND_URL="http://localhost:8082"
```

Then, as one example, replace every backend URL in your frontend code from:

```
` http://localhost:8082/api/books/${id}`
```

To:

```
process.env.NEXT_PUBLIC_BACKEND_URL + `/api/books/${id}`
```

When you need to change the backend URL, you can simply change the one line of code in the .env file or configure it in the deployment platform (depends on how you deploy it).

Updating the CSS file

While the functions are finished, the web app looks a bit bland. Let's add some flavour via the CSS. Edit the src/app/globals.css:

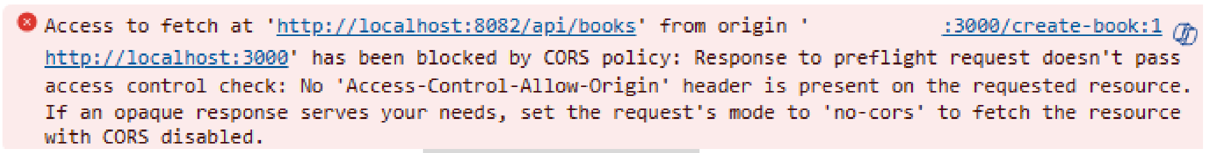
```
@tailwind base;
@tailwind components;
@tailwind utilities;

:root {
  --foreground-rgb: 0, 0, 0;
  --background-start-rgb: 214, 219, 220;
  --background-end-rgb: 255, 255, 255;
}

@media (prefers-color-scheme: dark) {
  :root {
    --foreground-rgb: 255, 255, 255;
    --background-start-rgb: 0, 0, 0;
    --background-end-rgb: 0, 0, 0;
  }
}

body {
  color: rgb(var(--foreground-rgb));
  background: linear-gradient(
      to bottom,
      transparent,
      rgb(var(--background-end-rgb))
    )
    rgb(var(--background-start-rgb));
}

@layer utilities {
  .text-balance {
    text-wrap: balance;
  }
}

.collapse.show {
  visibility: visible;
}

.CreateBook {
  background-color: #2c3e50;
  min-height: 100vh;
  color: white;
```

```css
}

.ShowBookDetails {
  background-color: #2c3e50;
  min-height: 100vh;
  color: white;
}

.UpdateBookInfo {
  background-color: #2c3e50;
  min-height: 100vh;
  color: white;
}

.ShowBookList {
  background-color: #2c3e50;
  height: 100%;
  width: 100%;
  min-height: 100vh;
  min-width: 100px;
  color: white;
}

/* BookList Styles */
.list {
  display: grid;
  margin: 20px 0 50px 0;
  grid-template-columns: repeat(4, 1fr);
  grid-auto-rows: 1fr;
  grid-gap: 2em;
}

.card-container {
  width: 250px;
  border: 1px solid rgba(0,0,.125);
  margin: 0 auto;
  border-radius: 5px;
  overflow: hidden;
}

.desc {
  height: 130px;
  padding: 10px;
}

.desc h2 {
  font-size: 1em;
  font-weight: 400;
}

.desc h3, p {
  font-weight: 300;
}
```

```
.desc h3 {
  color: #6c757d;
  font-size: 1em;
  padding: 10px 0 10px 0;
}
```

Running the app

Now we can run our frontend by `$ npm run dev` and backend by `$ npm run start`.
Once both parts are running, try to add a book, and observe the result.
If you follow everything in this tutorial, then there will be… nothing happens. Why? Is the backend URL wrong? Well, it is because of the CORS (and there is a clearer explanation). Your frontend and backend have different origins, so the access has been blocked. You can see the error in your browser console:



```
Access to fetch at 'http://localhost:8082/api/books' from origin '          :3000/create-book:1
http://localhost:3000' has been blocked by CORS policy: Response to preflight request doesn't pass
access control check: No 'Access-Control-Allow-Origin' header is present on the requested resource.
If an opaque response serves your needs, set the request's mode to 'no-cors' to fetch the resource
with CORS disabled.
```

It is not hard to fix. Go to your `backend/src/main.ts` and edit the code:

```
import { NestFactory } from '@nestjs/core';
import { AppModule } from './app.module';

async function bootstrap() {
  const app = await NestFactory.create(AppModule);
  // enable cors
  app.enableCors({ origin: true, credentials: true });
  const port = process.env.PORT || 8082;
  await app.listen(port, () => console.log(`Server running on port ${port}`));
}
bootstrap();
```

You can also do extra configuration of CORS, as mentioned in the document.
Now restart your backend and try it again, it should work (otherwise there may be other issues in your code, such as a typo of the URL).

Name: **Date:**

Worksheet Evidence:

This worksheet requires some answers to questions and **at least three selectively captured screenshots with an in depth reflection** as evidence. The aim is to be able to learn from the exercise, and evidence that.

For each of your three selected screenshots (or sequence of shots) in a brief paragraph or two reflect on:

- Why you have selected it?
- What have you learnt in this part of the worksheet?
- What was new or surprising?
- What useful external resource(s) did you consult and why? Provide a link(s) to the resource.

Evidence to Be uploaded to Canvas as well as Checked off by the Tutor (ideally by Week 4)

Evidence	Check
1. You could explain the critical steps of the MNNN stack tutorial that creates the Book app. You could include some written notes beside chosen screenshots that explain what is being done and anything new you learned or any tricks or mistakes you made.	
2. You could discuss the application components and how they relate. A screenshot(s) *of GitHub – code for frontend and backend; screen shot of the package.json file;* to support your answer may be suitable.	
3. You could discuss the data components and provide a screenshot of the MongoDB database created.	
4. What react library options are there to create each of the following: a. **Drop down widget** (such as needed to select the SE practice in the SPEED app) b. An **input form** (such as the article submission form needed for the SPEED app. c. A data **display table** (such as needed to display the evidence level and claims for the SPEED app).	
5. You could discuss the Books app and how it relates to the development of the SPEED product, and perhaps provide a screenshot of the app running locally on a browser.	
6. Explain the purpose of the CORS command.	

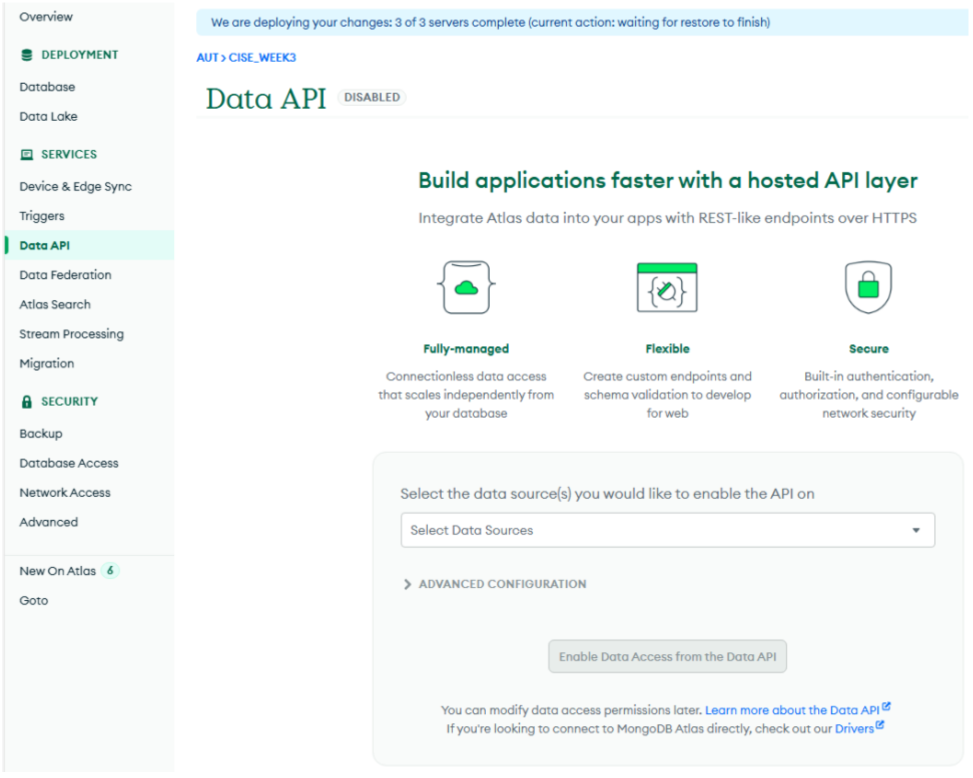
Appendix

While we have used local data in this worksheet, when running the books app, the additional material below may be used for later reference for importing data, when populating the MongoDB database for your team project.

Import data into MongoDB Atlas via Data API

1. Enable Data API

   Go to the "Data API" page, select your cluster (cluster0 by default, or the name you set) and enable the Data API.



We are deploying your changes: 3 of 3 servers complete (current action: waiting for restore to finish)

Data API DISABLED

Build applications faster with a hosted API layer

Integrate Atlas data into your apps with REST-like endpoints over HTTPS

Fully-managed Connectionless data access that scales independently from your database

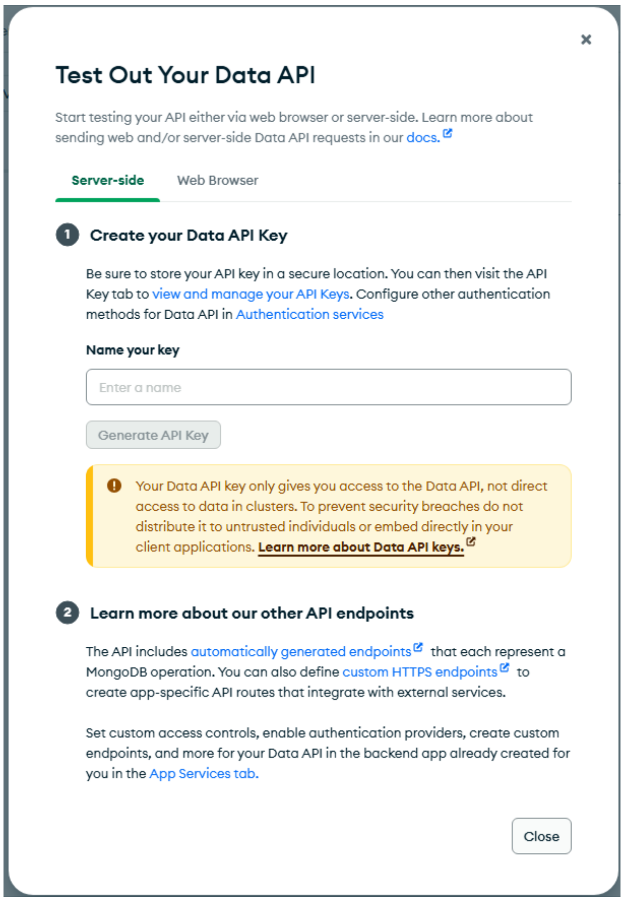
Flexible Create custom endpoints and schema validation to develop for web

Secure Built-in authentication, authorization, and configurable network security

Select the data source(s) you would like to enable the API on

You can modify data access permissions later. Learn more about the Data API
If you're looking to connect to MongoDB Atlas directly, check out our Drivers

A "Test Out Your Data API" page will pop out.



×

Test Out Your Data API

Start testing your API either via web browser or server-side. Learn more about sending web and/or server-side Data API requests in our docs.

Server-side | Web Browser

1 Create your Data API Key

Be sure to store your API key in a secure location. You can then visit the API Key tab to view and manage your API Keys. Configure other authentication methods for Data API in Authentication services

Name your key

Enter a name

Generate API Key

Your Data API key only gives you access to the Data API, not direct access to data in clusters. To prevent security breaches do not distribute it to untrusted individuals or embed directly in your client applications. Learn more about Data API keys.

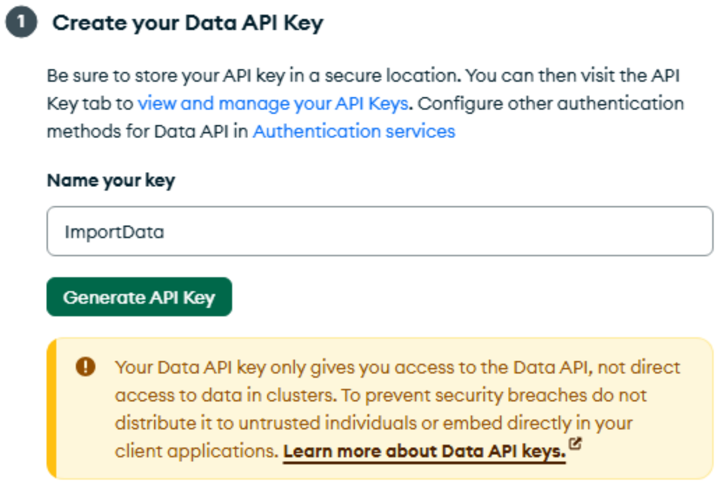
2 Learn more about our other API endpoints

The API includes automatically generated endpoints that each represent a MongoDB operation. You can also define custom HTTPS endpoints to create app-specific API routes that integrate with external services.

Set custom access controls, enable authentication providers, create custom endpoints, and more for your Data API in the backend app already created for you in the App Services tab.

Close

We need to create an API key for API access. Use whatever name you like.



1 Create your Data API Key

Be sure to store your API key in a secure location. You can then visit the API Key tab to view and manage your API Keys. Configure other authentication methods for Data API in Authentication services

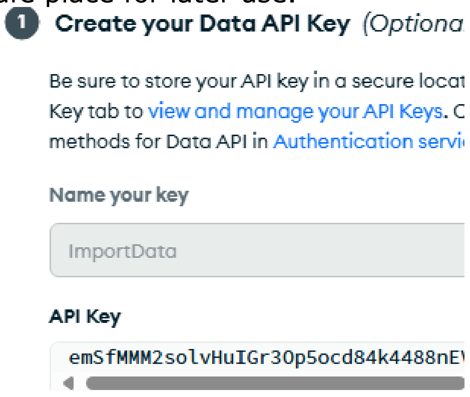
Name your key

ImportData

Generate API Key

Your Data API key only gives you access to the Data API, not direct access to data in clusters. To prevent security breaches do not distribute it to untrusted individuals or embed directly in your client applications. Learn more about Data API keys.

Hit "Generate API Key" and wait for a short while, you will get your API key. Save it in a safe place for later use.



1 Create your Data API Key (Optiona

Be sure to store your API key in a secure locat
Key tab to view and manage your API Keys. C
methods for Data API in Authentication servi

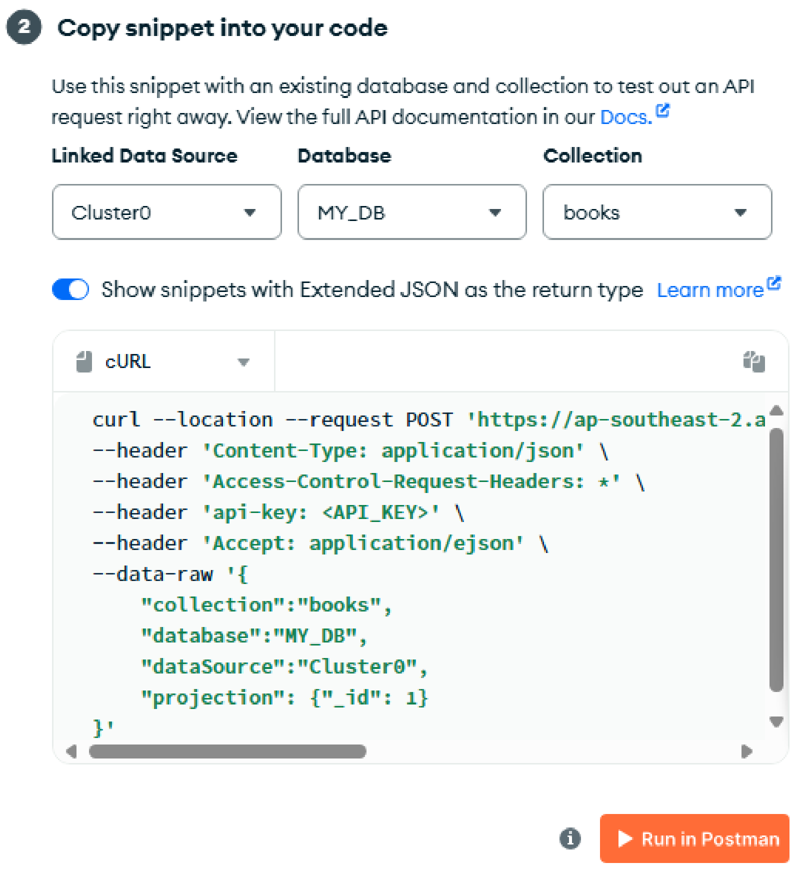
Name your key

ImportData

API Key

```
emSfMMM2solvHuIGr30p5ocd84k4488nE
```

Then you can select the data source, database and collection you want to access, the corresponding access information will be generated for you:



2 Copy snippet into your code

Use this snippet with an existing database and collection to test out an API request right away. View the full API documentation in our Docs.

Linked Data Source	Database	Collection
Cluster0	MY_DB	books

Show snippets with Extended JSON as the return type Learn more

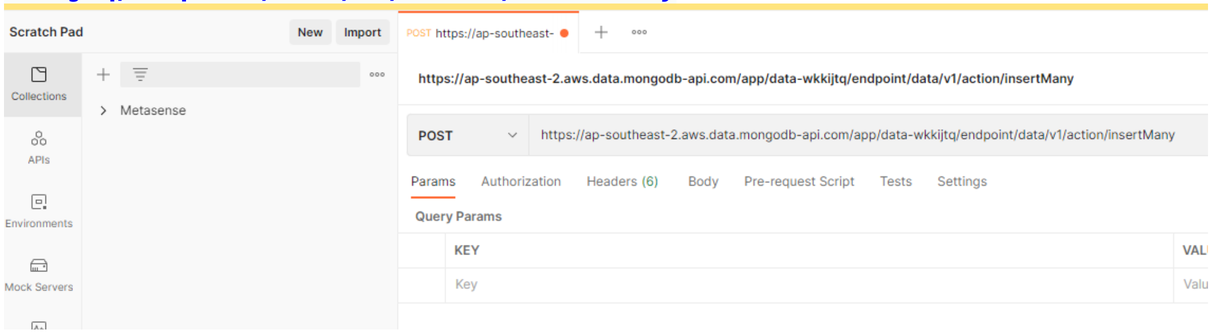
cURL

```
curl --location --request POST 'https://ap-southeast-2.a
--header 'Content-Type: application/json' \
--header 'Access-Control-Request-Headers: *' \
--header 'api-key: <API_KEY>' \
--header 'Accept: application/ejson' \
--data-raw '{
    "collection":"books",
    "database":"MY_DB",
    "dataSource":"Cluster0",
    "projection": {"_id": 1}
}'
```

Record the endpoint url (the url after POST in the first line), "collection", "database" and "dataSource" for later use.

2. Import data into database
   You can use curl tool, or Postman (which I recommend).
   Open Postman, create a new request, choose POST method, and input the url endpoint you get from previous step:
   https://ap-southeast-2.aws.data.mongodb-api.com/app/data-wkkijtq/endpoint/data/v1/action/findOne (This is my link; you need to use yours)

   Change the "findOne" to "insertMany", since we are going to insert (multiple) data:
   https://ap-southeast-2.aws.data.mongodb-api.com/app/data-wkkijtq/endpoint/data/v1/action/insertMany

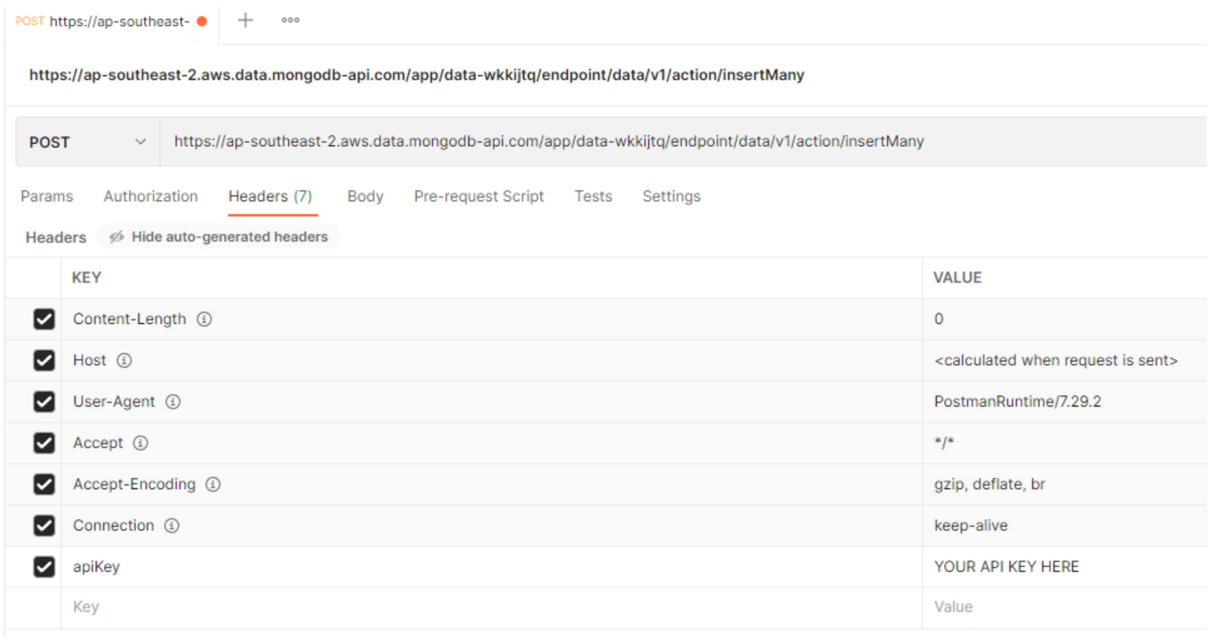


https://ap-southeast-2.aws.data.mongodb-api.com/app/data-wkkijtq/endpoint/data/v1/action/insertMany

POST https://ap-southeast-2.aws.data.mongodb-api.com/app/data-wkkijtq/endpoint/data/v1/action/insertMany

KEY	VAL
Key	Valu

Go to the Headers tab, add a new item "apiKey", put your key here.


POST https://ap-southeast-

https://ap-southeast-2.aws.data.mongodb-api.com/app/data-wkkijtq/endpoint/data/v1/action/insertMany

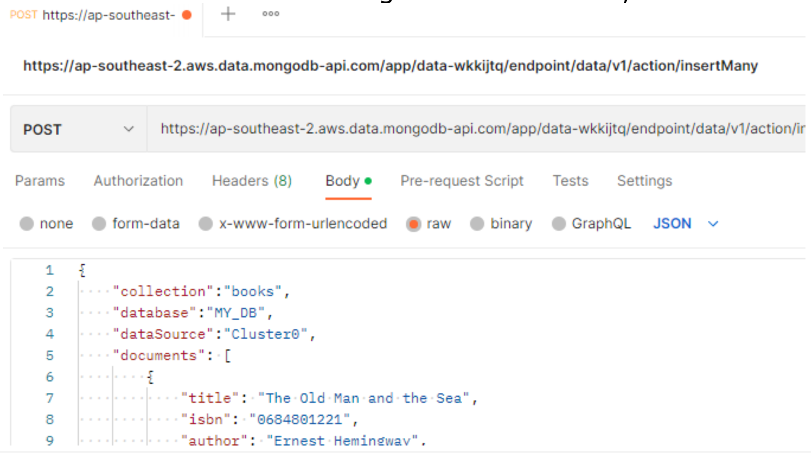
POST https://ap-southeast-2.aws.data.mongodb-api.com/app/data-wkkijtq/endpoint/data/v1/action/insertMany

Params | Authorization | Headers (7) | Body | Pre-request Script | Tests | Settings

Headers — Hide auto-generated headers

	KEY	VALUE
✓	Content-Length	0
✓	Host	<calculated when request is sent>
✓	User-Agent	PostmanRuntime/7.29.2
✓	Accept	*/*
✓	Accept-Encoding	gzip, deflate, br
✓	Connection	keep-alive
✓	apiKey	YOUR API KEY HERE
	Key	Value

Then go to the Body tab, select "raw" and "JSON", and copy and paste the following content. Remember to change the "collection", "database" and "dataSource" to yours.


POST https://ap-southeast-

https://ap-southeast-2.aws.data.mongodb-api.com/app/data-wkkijtq/endpoint/data/v1/action/insertMany

POST https://ap-southeast-2.aws.data.mongodb-api.com/app/data-wkkijtq/endpoint/data/v1/action/ir

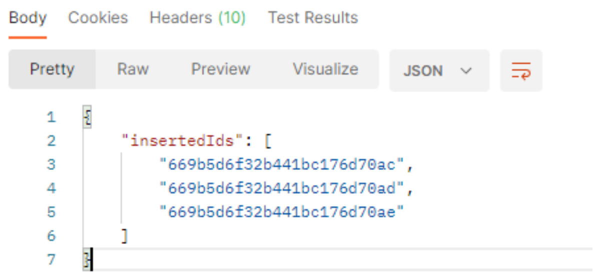
Params | Authorization | Headers (8) | Body ● | Pre-request Script | Tests | Settings

none | form-data | x-www-form-urlencoded | [✓ raw] | binary | GraphQL | JSON

```
{
    "collection":"books",
    "database":"MY_DB",
    "dataSource":"Cluster0",
    "documents": [
        {
            "title": "The Old Man and the Sea",
            "isbn": "0684801221",
            "author": "Ernest Hemingway",
```

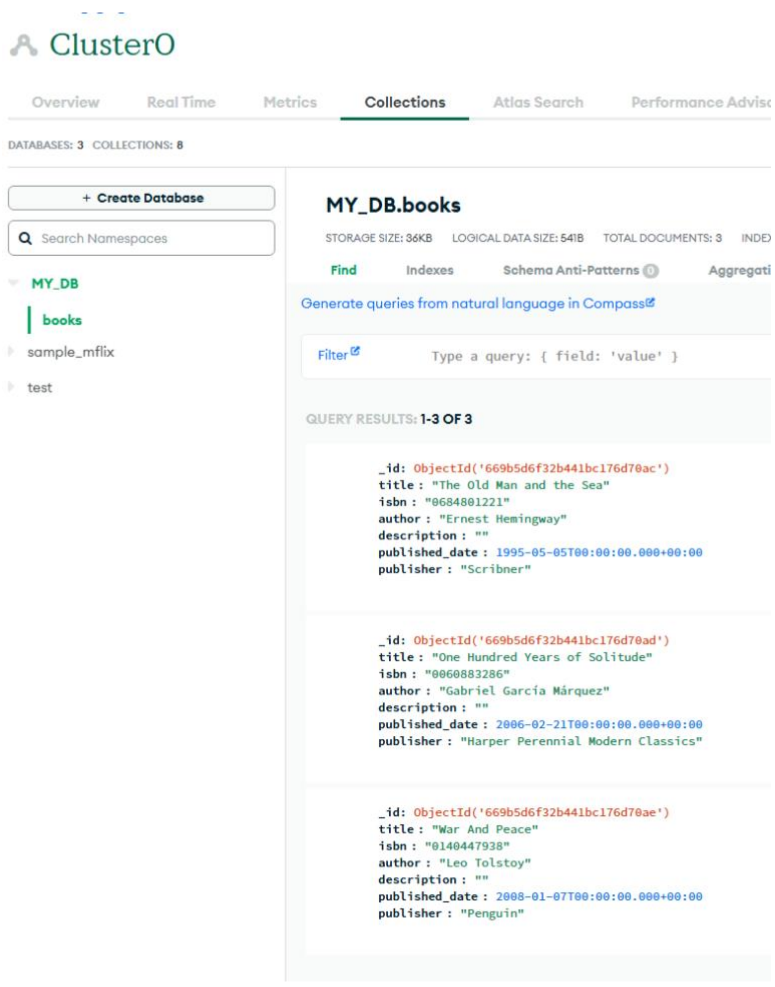
```
{
    "collection":"books",
    "database":"MY_DB",
    "dataSource":"Cluster0",
    "documents": [
        {
            "title": "The Old Man and the Sea",
            "isbn": "0684801221",
            "author": "Ernest Hemingway",
            "description": "",
            "published_date": {
                "$date": {
                    "$numberLong": "799632000000"
                }
            },
            "publisher": "Scribner"
        },
        {
            "title": "One Hundred Years of Solitude",
            "isbn": "0060883286",
            "author": "Gabriel García Márquez",
            "description": "",
            "published_date": {
                "$date": {
                    "$numberLong": "1140480000000"
                }
            },
            "publisher": "Harper Perennial Modern Classics"
        },
        {
            "title": "War And Peace",
            "isbn": "0140447938",
            "author": "Leo Tolstoy",
            "description": "",
            "published_date": {
                "$date": {
                    "$numberLong": "1199664000000"
                }
            },
            "publisher": "Penguin"
        }
    ]
}
```

Hit the “Send” button, you should get this result (ids can be different) if everything is correct:



```
{
    "insertedIds": [
        "669b5d6f32b441bc176d70ac",
        "669b5d6f32b441bc176d70ad",
        "669b5d6f32b441bc176d70ae"
    ]
}
```

Check your database collection, you should see the three sample data:



QUERY RESULTS: 1-3 OF 3

```
_id: ObjectId('669b5d6f32b441bc176d70ac')
title : "The Old Man and the Sea"
isbn : "0684801221"
author : "Ernest Hemingway"
description : ""
published_date : 1995-05-05T00:00:00.000+00:00
publisher : "Scribner"
```

```
_id: ObjectId('669b5d6f32b441bc176d70ad')
title : "One Hundred Years of Solitude"
isbn : "0060883286"
author : "Gabriel García Márquez"
description : ""
published_date : 2006-02-21T00:00:00.000+00:00
publisher : "Harper Perennial Modern Classics"
```

```
_id: ObjectId('669b5d6f32b441bc176d70ae')
title : "War And Peace"
isbn : "0140447938"
author : "Leo Tolstoy"
description : ""
published_date : 2008-01-07T00:00:00.000+00:00
publisher : "Penguin"
```