MERN Tutorial

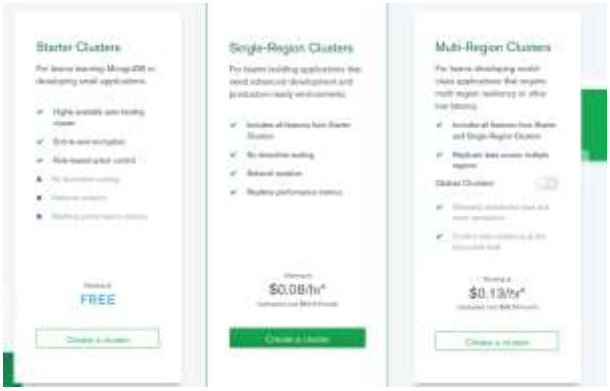
By the end of this document, you will have:

1. deployed your first MERN application to Heroku.
2. understood to some extent how the app's code is organized.
3. installed and used Postman. (optional)

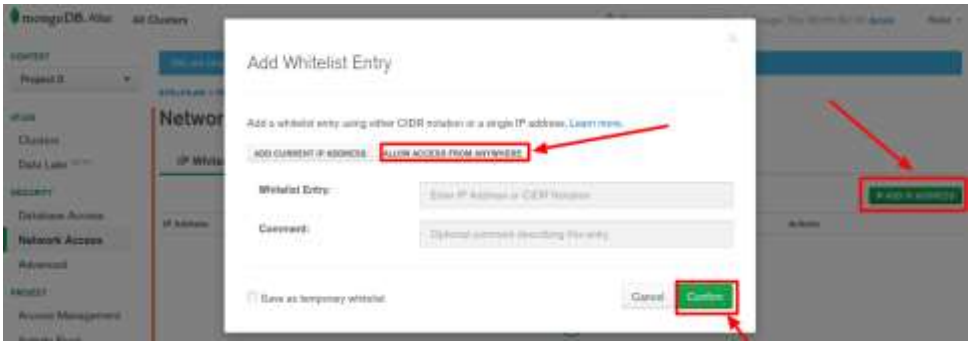
Section 1: Instructions to Deploy

By the end of this section, you will have successfully deployed this website.

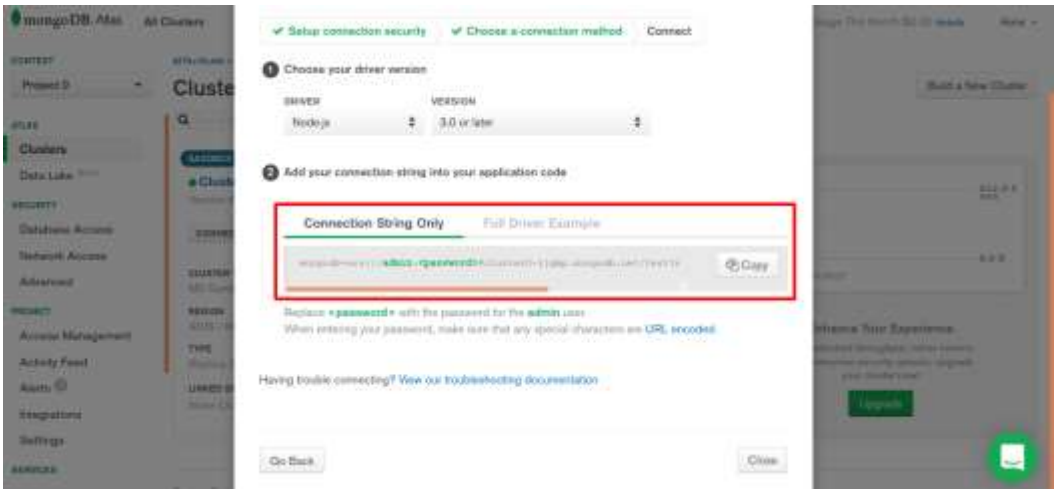
1. Login to GitHub and fork this repository.
2. Our app uses Mongo as its database. We'll be using MongoDB Atlas to host the database. Create an account on MongoDB Atlas and sign in.
3. Create a cluster using the "Starter Clusters" or free category. (left-most)



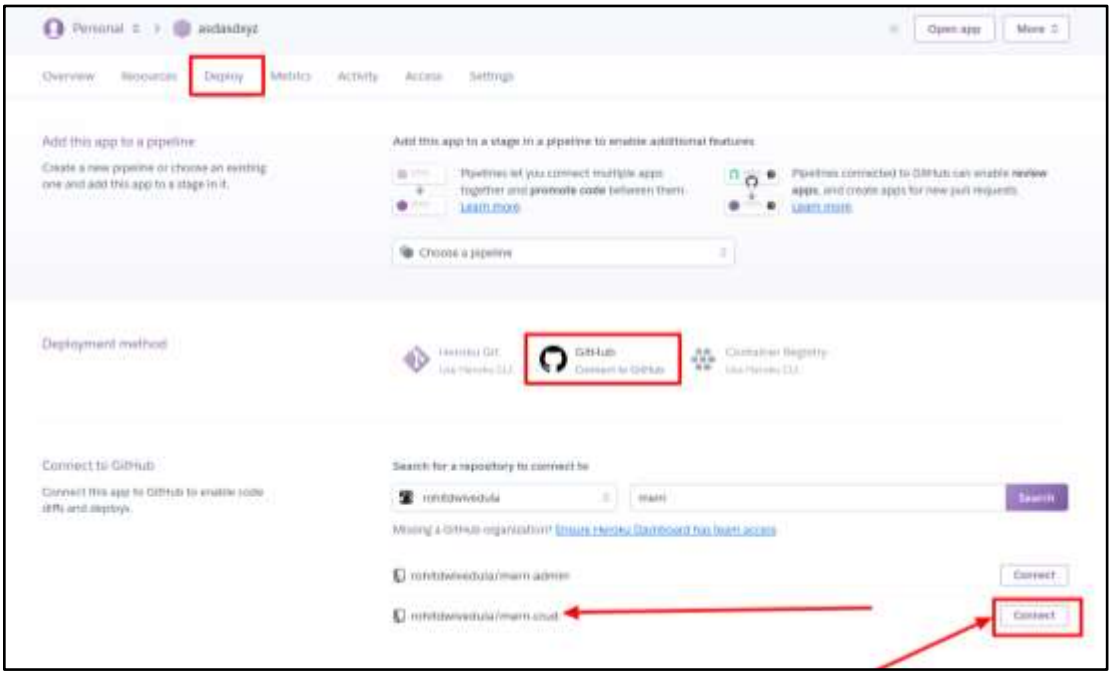
4. In the next page, you'll be prompted to choose the configuration of the server - don't change any of the default options except for the cloud service region - make that "Mumbai/India". It might take a minute or two for the cluster to be created.
5. On the right-hand side toolbar, click on the "Network Access" option (under 'Security'). Click on the "Add IP Address" button and whitelist all IPs for now. [this is NOT a good security practice, but it will simplify your life, so just use this for now]



6. Go to the “Database Access” menu and create a new user. Remember the login and password you used. Once this is done go back to the Clusters screen and click on connect.
7. Since we’re using a web app, we want to “Connect with the application” and copy the connector code. Replace the “<password>” in the connector with the password you set in the earlier step. Replace the “<>”This is the **URI** that you will use to connect to your database.

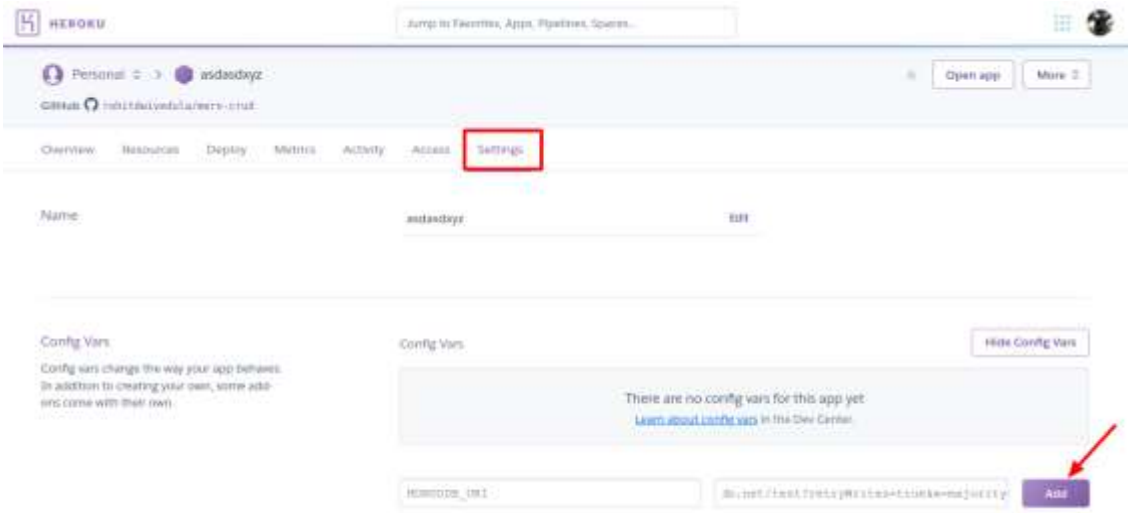


8. Now that we have a database up and running, sign in to Heroku and create a new app. The new app’s dashboard will look something like shown below. Click on the deploy tab and click on the deploy using GitHub option. You will be asked to log in to your GitHub account and give access, after which you’ll be able to find the repo that you forked in step 1. Click on the connect button.



9. Scroll down on the same page a bit more to find the “manual deploy” option and deploy the master branch.

10. While the app is being deployed go to the Settings tab, and add a config variable called ‘MONGODB_URI’ with the value being the database connection URI that you created in step 7.



11. Click on “open app” once it’s been deployed. Congratulations! You’ve deployed your first MERN website! :)

Section 2: Understanding What's Happening [ExpressJS]

The web app has two parts: the front-end and the backend. The backend communicates with the database and contains most of the business logic. The front end is the web app that you see in your browser - the design, the buttons and all.

The directory structure looks a bit like this:

```
├── index.js
├── models
│   ├── Movie.js
├── movies-crud
│   ├── build
│   ├── node_modules
│   ├── package.json
│   ├── public
│   ├── README.md
│   ├── src
│   └── yarn.lock
├── node_modules
├── package.json
├── package-lock.json
```

Outline:

1. The backend server has only two files - `index.js` and `models/Movie.js`.
2. `index.js` sets up the Express server and defines the routes (what URLs are available and what each one does)
3. The `models/Movie.js` file contains the *schema* of the MongoDB database. A schema is a description of how the data is organised in the database.[1]
4. The directory `movies-crud` contains the front end components (ReactJS)
5. The `node_modules` folders contain the libraries required for running the apps.
6. The package.json file contains information that the Node Package Manager (npm) uses to manage the dependencies of a project. We see there are two package.json files in the directory structure - one each for the front end and the back end.[2]

Both the `index.js` file and the `models/Movie.js` file have detailed comments explaining what each part of the code does. Go through these files to see how the backend code works.

[1] Read the documentation for Mongoose here.
[2] What is the file package.json?

Section 3: Using Postman

Postman is a tool that lets you send HTTP requests to a server. As you've seen in the previous section, the backend code defines *endpoints* (or URLs), each of which has a specific function. In the web app, the front end React code uses these endpoints to provide the functionality to the user. In this section, we'll use Postman to test the backend code. Postman can be downloaded from here.

Install the Node Server On Your Laptop

Clone the server on your computer using these commands:[3]

```
git clone github.com/rohitdwivedula/mern-crud
cd mern-crud
cp .env-copy .env
```

The last command above creates a file called `.env`. This file stores *environment variables* that the program needs. Open the `.env` file with a text editor and edit the MongoDB URI in the file [the one that you created on MongoDB Atlas in Step 7 during deployment of your app]. Then run the commands below to install and run the server.

```
npm install
cd client
yarn install
yarn build
cd ..
nodemon index.js
```

If last command does not work, then please try running `npx nodemon index.js`. You should get a message that says something along the lines of `Listening on port 5050.`

Note: The MongoDB Atlas connection might not work on campus internet. If you see a MongoDB timed out, or similar error in the last step, try connecting your PC/laptop to mobile hotspot instead.

[3] `nodemon` is a program that monitors the files and automatically restarts the node application when file changes in the directory are detected. While you could use `node index.js` to run the app, you'd have to manually stop and start the app after each change - `nodemon` does this for you.

Open a web browser, and go to `localhost:5050` - you'll see the website open. In the next section, we'll use Postman to add a new movie to the database.[4] Postman is a very useful tool for debugging API endpoints.

Using Postman

Try adding the movie "A Beautiful Mind" to the database. We see that the `/api/movie/add` endpoint can be used for adding a movie and this is the format to send the movie data:

```
{
    "title": "A Beautiful Mind",
    "year": 2001,
    "desc": "John Nash, a brilliant but asocial mathematical genius,
suffers from paranoid schizophrenia. He ultimately overcomes his ailment
and emerges free of any trauma.",
    "cast": ["Russell Crowe", "Ed Harris", "Judd Hirsch"]
}
```

A screenshot of the Postman app

[4] This, this and this are decent resources to get started with using Postman. Fun fact: The founder and CEO of Postman is a BITS alumnus.

The response indicates that the movie was inserted into the database. The response also indicated that the **id** of the movie in the database is `5e2f3d184f9e4146c6a8e053`. The identifier you get will vary from system to system.

You might have noticed that while the website has the functionality to add, view and remove movies, there's no way to edit them. However, if you've gone through the code in `index.js`, you'd have noticed there's an endpoint `/api/movie/:id/update` that allows you to update the details of the movie. Try to use Postman to update the movie that we just inserted. Replace the ":id" in the URL with the id of the movie that you want to edit. The body of the POST request should contain the attributes you want to change. For example, sending this body:

```
{
"title": "A Beautiful Mind (English)"
}
```

to `localhost:5050/api/movie/5e2f3d184f9e4146c6a8e053/update` will edit the name of the movie from "A Beautiful Mind" to "A Beautiful Mind (English)". Try playing around with Postman and the app to add, delete, view and edit movies to get comfortable using it.

Queries

For any queries about this document you may contact the TA for the course, Rohit Dwivedula (2017A7PS0029H): f20170029@hyderabad.bits-pilani.ac.in