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

Creating a Dashboard

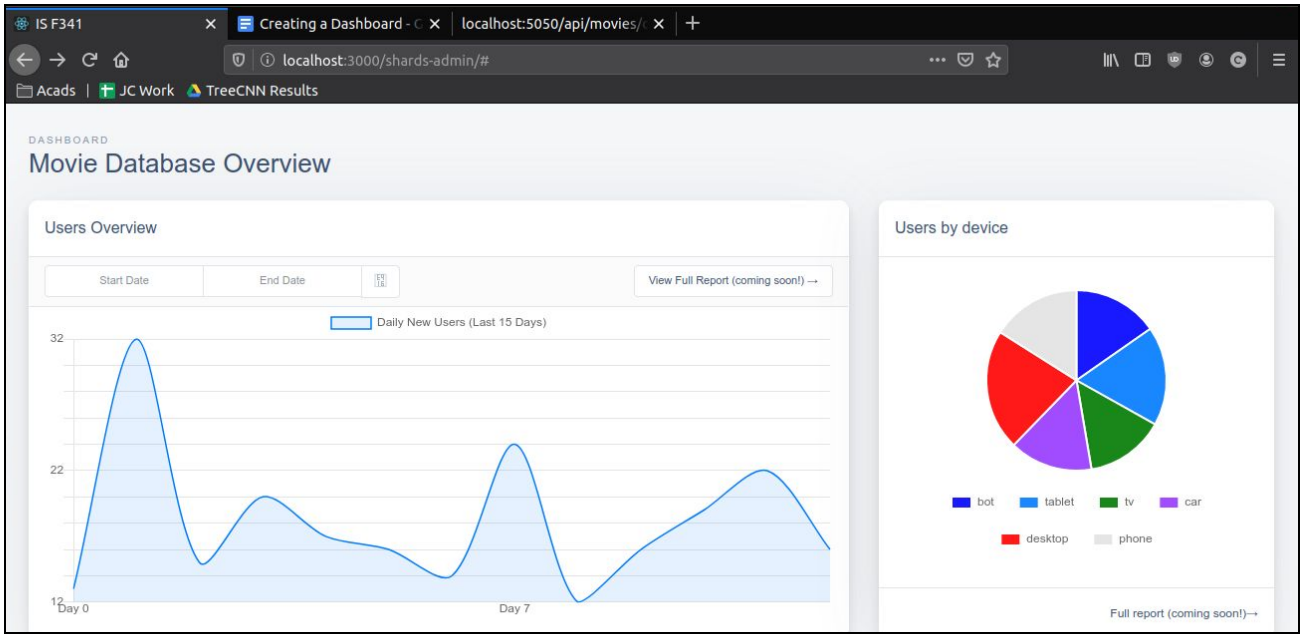
In this tutorial, we'll be using two readymade ReactJS template libraries to make an admin dashboard for our web application. Our admin dashboard will be relatively simple - it'll show the number of movies and users, keep track of the number of hits our API is getting and maybe plot the number of movies and users over time. We'll also be generating some mock data using Mockaroo so that we can populate our data with realistic-looking data.

To make the admin dashboard work, we'll have to make some changes to our backend API endpoints and add a couple of endpoints too. We'll display the data in the Shards Dashboard Lite. There are tons of free ReactJS templates online, and this link might be a good place to start your search. The aim of this exercise is to partly make you comfortable using someone else's code/templates and build upon it. Regarding what dashboard template to choose - that's largely something you'll have to decide on your own, based on what you think would present data the best for your application.

To be specific, this is the data we will try to report in the dashboard:

1. the total number of users and movies currently stored by the app
2. number of movies and users in the application over time.
3. number of hits on each of the API endpoint, both totals and day-wise aggregates

We will continue to use the same Github repo that's been used in earlier tutorials. The end-goal will be a one-page simple dashboard that looks something like this.



Adding Backend Functionality

Adding endpoints for getting total number of users and movies.
You'll have to add endpoints that look something like this to get the total count of all movies/users/whatever.

```
app.get("/api/movies/count", (req,res)=>{
    /// get number of movies in the database
    Movie.countDocuments({},(err, count) => {
        if (err) {
       res.status(500).send({status:false, error:err})
        }
        else {
       res.status(200).send({"movies": count});
        }
       }
   );
});
```

Tracking Number of Movies and Users Over Time
Showing a graph like the one shown in the previous page, reporting the daily new users, can be helpful from a business perspective, helping admins of the app gain insights into user trends. To show graphs like these, we need to add a small code snippet to our Mongoose model - telling it to save the time an object was created in the document. This can be achieved through Mongoose timestamps. The syntax for this is shown below.[1]

```
var MySchema = new Schema(
  {
      attr1: {type: String},
      attr2: {type: Number}
      /// define your Mongoose schema here
  },
  {
      timestamps: {
          createdAt: 'created_at',
          updatedAt: 'updated_at'
      }
  }
);
```

[1] This StackOverflow thread claims that you do not need a created_at attribute in a MongoDB object since that can just be calculated from the object _id.

Recording Number of API Hits

Another useful thing to track in the dashboard might be *user engagement* or how much our users are engaging with the application. One way to measure this would be through the number of times they visit our page. To keep track of the users, we create a PageView schema which stores two attributes: the user's email, and the API endpoint that was hit by the user (all movies, delete movies, etc) to keep track of what functionalities users are using, along with a timestamp.

```
var PageViewSchema = new Schema({
    email: {
      type: String,
      required: true
      },
       endpoint: {
             type: String,
             required: true
       },
       device: {
             type: String,
             required: true
       }
})
```

Take a look at the page_view.js file in the routes/api directory to see how it's working in detail.

```
record_activity: function (user, api_endpoint, device)
    {
       // record that a user has accessed an endpoint right now.
       // make sure to add a Privacy Policy to your app :p
       var record = {email: user, endpoint: api_endpoint, device_type: device};
       PageView.create(record,(err, obj) => {
             if (err) {
                   throw err;
             }
             else {
                   console.log("State surveillance successful.");
             }
       });
    }
```

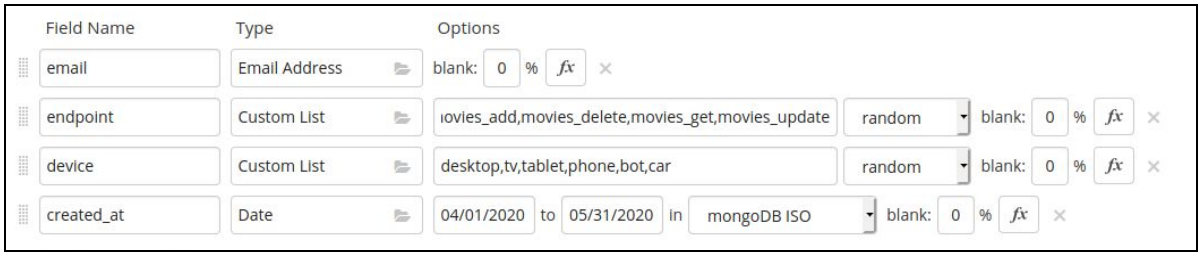
Now that we have all this data being stored in the database we need to define endpoints to

retrieve this data. These endpoints are defined in routes/api/stats.js. To understand what's happening there, take a look at the documentation for $match and aggregate in the Mongoose docs and the MongoDB manual[2].

Creating Mock Data

You can use Mockaroo to create custom data, and then export it to CSV. The free version of Mockaroo lets you create up to 1000 rows of fake data without login, which should be enough for our needs. The data I generated can be found in the mock_data folder in the repository. After downloading the CSV, insert all the data generated into your database. A script insert_data.js has been provided in the same directory as the mock data as a sample[3]. You could probably make edits to the script, based on your data and your MongoDB URI and use it.

When generating mock data for your application if you have any *Date* attribute, make sure to select the date format as "mongoDB ISO" in Mockaroo. This is the configuration used in my application to generate mock data for the PageView model:



After downloading the CSV, you'll notice that the timestamp is formatted inside multiple brackets - remove all of them (find and replace) as shown below before putting it through the insert_data.js script.

```
email,endpoint,device,created_at
cbodechon0@arstechnica.com,movies_all,desktop,"{""$date"":""2020-05-03T04:10:23.000Z""}"
gkenney1@hexun.com,movies_delete,tv,"{""$date"":""2020-04-06T03:34:41.000Z""}"
lmorby2@com.com,movies_update,car,"{""$date"":""2020-05-30T10:06:53.000Z""}"
mesherwood3@xinhuanet.com,movies_update,tablet,"{""$date"":""2020-05-10T21:22:29.000Z""}"
dlelievre4@plala.or.jp,movies_update,phone,"{""$date"":""2020-04-06T08:01:11.000Z""}"
oholbie5@nasa.gov,movies_get,tablet,"{""$date"":""2020-05-06T17:06:06.000Z""}"
```

Above: original data and below: data to send into script.

```
email,endpoint,device,created_at
cbodechon0@arstechnica.com,movies_all,desktop,2020-05-03T04:10:23.000Z
gkenney1@hexun.com,movies_delete,tv,2020-04-06T03:34:41.000Z
lmorby2@com.com,movies_update,car,2020-05-30T10:06:53.000Z
mesherwood3@xinhuanet.com,movies_update,tablet,2020-05-10T21:22:29.000Z
dlelievre4@plala.or.jp,movies_update,phone,2020-04-06T08:01:11.000Z
```

[2] This StackOverflow thread discusses the same thing - your dates must be formatted as ISODates() inside MongoDB for comparision operators in MongoDB to work.
[3] The script uses the csv-parser package.

Integrating with the React Dashboards

Shards Dashboard

The Shards Dashboard can be downloaded from here. After downloading the dashboard you want to work with, put the downloaded zip file inside the src directory of your client. The first thing you'll want to do is remove most of the components that are available in the React package, and only save those items that you need. Once this is done, you'll have to add the API calls to all the endpoints you've built so far. To see how one sample API request integration with the shards dashboard looks like, take a look at these file in the Git repo, which show how the two graphs shown in the first page of this tutorial are made:

mern-crud/client/src/shards-dashboard/src/components/blog/UsersOverview.js
mern-crud/client/src/shards-dashboard/src/components/blog/UsersByDevice.js

However, this specific example is unlikely to help you much - each React dashboard is structured differently and so are your API endpoints and the data you want to show. In general, some tips that might help you are:

1. Once you find a specific react element that you want to change dynamically by integrating with the API, the functions componentDidMount() or componentWillMount() are good places to start - you can organise your API calls there and save the results as a class variable.
2. All the React templates come with dummy data - oftentimes, the only change you'll have to make is replacing their static array of values with a variable containing the data you got from the API endpoint. The only change required in the code samples shown below was changing the array of values [2, 3, 4, ...] with a variable, data. data contains the required array loaded from an API call.

```
var time_series_data = {
  labels: labels,
  datasets: [
    {
      label: "Daily New Users (Last 15 Days)",
      fill: "start",
      data: [2,3,4,5,6,4,2,1,2],
      backgroundColor: "rgba(0,123,255,0.1)",
      borderColor: "rgba(0,123,255,1)",
      pointBackgroundColor: "#ffffff",
      pointHoverBackgroundColor: "rgb(0,123,255)
      borderWidth: 1.5,
      pointRadius: 0,
      pointHoverRadius: 3
    }
  ]
};
```

```
var time_series_data = {
  labels: labels,
  datasets: [
    {
      label: "Daily New Users (Last 15 Days)",
      fill: "start",
      data: values,
      backgroundColor: "rgba(0,123,255,0.1)",
      borderColor: "rgba(0,123,255,1)",
      pointBackgroundColor: "#ffffff",
      pointHoverBackgroundColor: "rgb(0,123,255)",
      borderWidth: 1.5,
      pointRadius: 0,
      pointHoverRadius: 3
    }
  ]
};
```

For the most part - you'll have to figure out the integration of your app with the dashboard on your own. There are too many variables at play when using other people's code.