

# TIK TAK PART 2

Time  
Tracker

Login

Register

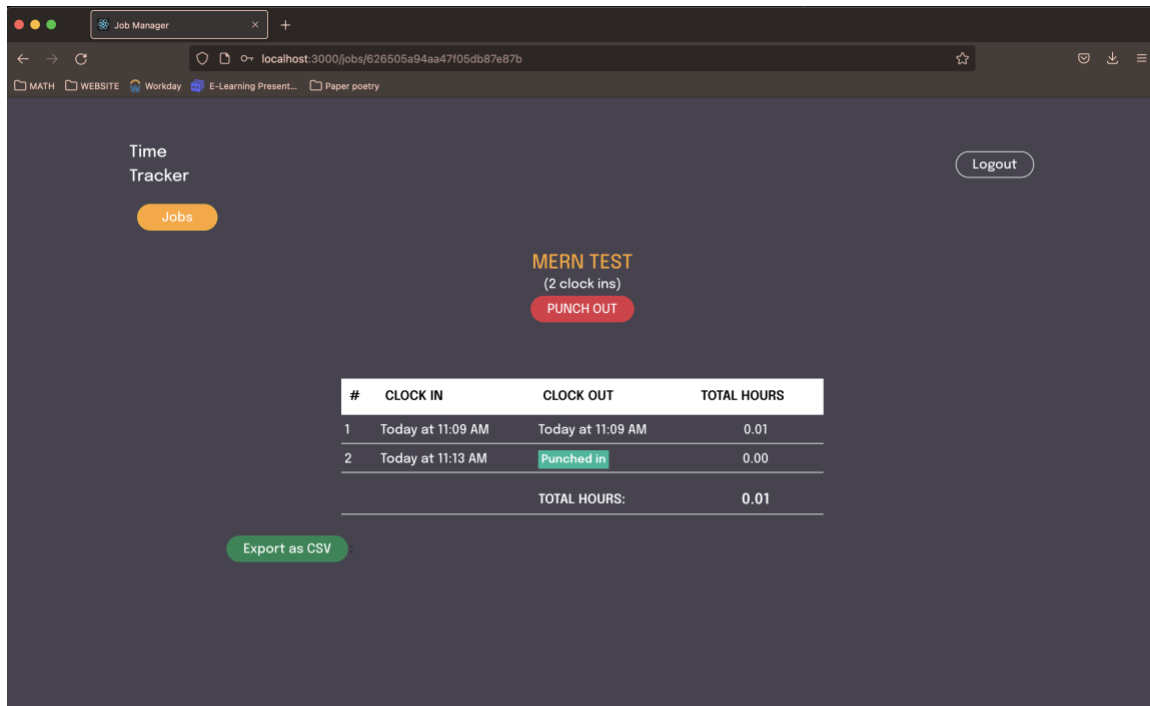


TIK  
TAK



### CSV Export Functionality

It would be very helpful for the users to have a functionality where they can export all the punches they have made for specific jobs. For example, you can see the button Export as CSV in the picture below:



Instead of the user having to open the website every time he wants to see the punches, or show them to create a report for the employer, by clicking the button “Export as CSV” he will

have a list with all the punches for that specific job. Here is how the CSV file looks from the inside:

AutoSave OFF

punches-2

Home Insert Draw Page Layout Formulas Data Review View Tell me

Paste Calibri (Body) 12 A+ A- B I U Wrap Text Merge & Center General Conditional Formatting Format as Table Cell Styles Insert Delete Format Sort & Filter Find & Select Analyze Data

Possible Data Loss Some features might be lost if you save this workbook in the comma-delimited (.csv) format. To preserve these features, save it in an Excel file format. Save As...

	A	B	C	D	E	F	G	H
1	id	clockin	clockout	total time				
2	6263ab7670f41d912b5edfbb	Yesterday at 10:32 AM	Yesterday at 10:33 AM	0.02				
3	6263baa3901e3d9f6c3e8657	Yesterday at 11:36 AM	Yesterday at 11:37 AM	0.01				
4	6263c32e71ee939fd19457bd	Yesterday at 12:13 PM	Yesterday at 12:14 PM	0.02				
5	6263c40071ee939fd19457ee	Yesterday at 12:16 PM	Yesterday at 12:17 PM	0.02				
6								
7	Total hours:			0.07				
8								
9								
10								
11								
12								
13								
14								
15								
16								
17								
18								
19								
20								
21								
22								

punches-2

Ready 173%

You can see that there are 4 field names in the csv file:

- Id of the punch (distinct identifier for our system)
- Clocking time
- Clock out time
- Total time spend punched in (the difference between clock out and clock in)

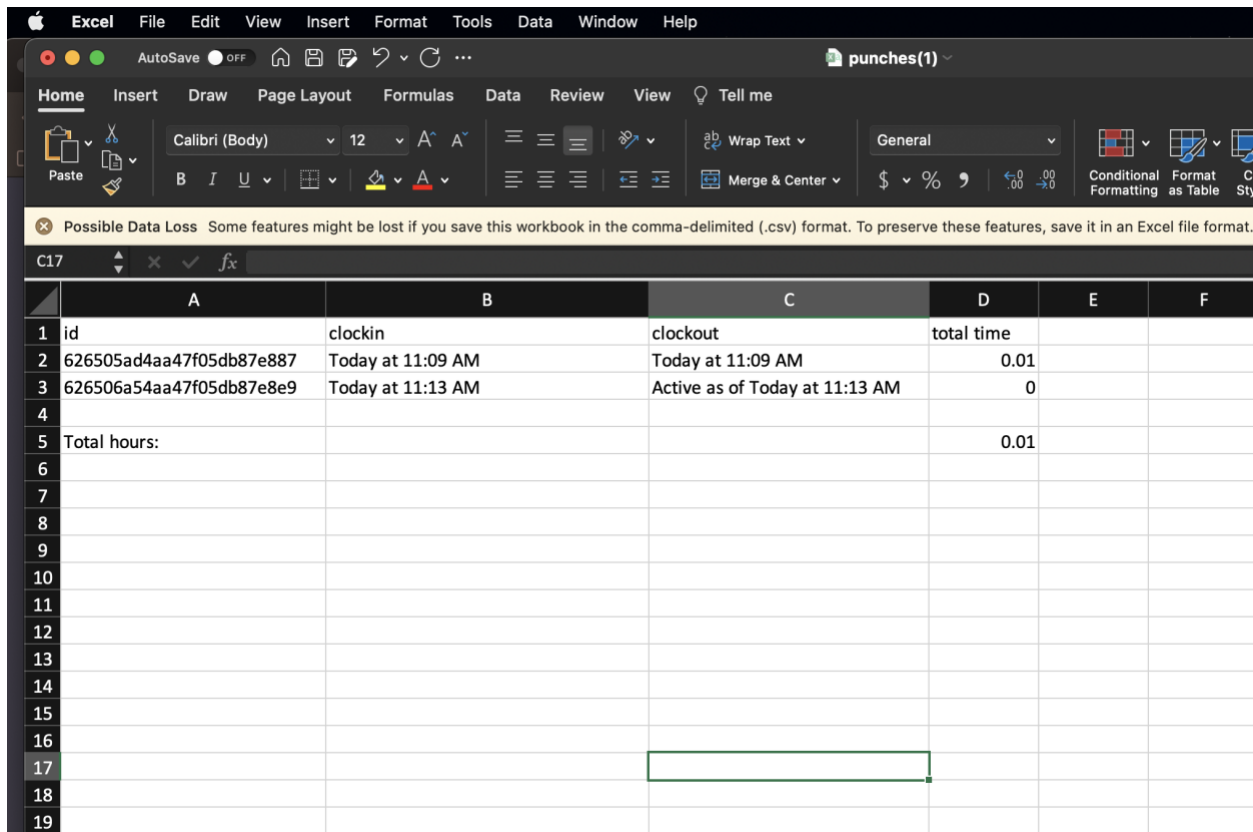
Also, just for reference and statistical purposes, I included a field total time, with the total time calculated from the sum of all punches.

If the user has an active punch when clicking the “Export as CSV button” then the field for that punch in the excel will look as the following:

If the user is currently punched in:

#	CLOCK IN	CLOCK OUT	TOTAL HOURS
1	Today at 11:09 AM	Today at 11:09 AM	0.01
2	Today at 11:13 AM	Punched in	0.00
TOTAL HOURS:			0.01

After clicking Export as CSV, the following CSV will be downloaded:



	A	B	C	D	E	F
1	id	clockin	clockout	total time		
2	626505ad4aa47f05db87e887	Today at 11:09 AM	Today at 11:09 AM	0.01		
3	626506a54aa47f05db87e8e9	Today at 11:13 AM	Active as of Today at 11:13 AM	0		
4						
5	Total hours:			0.01		
6						
7						
8						
9						
10						
11						
12						
13						
14						
15						
16						
17						
18						
19						

The cell (in our case) C3 will have the text “Active as of Today at 11:13 AM” – meaning that it is not punched out yet, thus this punch hours will not be calculated in the total amount.

For implementing this feature, I used a Javascript 3<sup>rd</sup> party library named “react-csv”. This feature implementation was very straightforward when following the documentation, which can be found here: <https://www.npmjs.com/package/react-csv>.

I created a button with the following code:

```
<CSVLink data={csv_data} filename={"punches.csv"}>
  <button className='btn btn-success'>
    Export as CSV
  </button>
</CSVLink>
```

The component `<CSVLink/>` is provided from react-csv library. Then, I used it to wrap my button with text Export as CSV. The filename attribute contains the filename for the exported csv, and the data attribute contains the data to be exported. I prepared this data while rendering the component by taking all the punches of current job from the database and giving them the structure needed for csv exporting. The format of the csv\_data array should look as the following:

- Should be and multidimensional array.
- The first array in the array should be the names of the columns.
  - ID
  - CLOCK IN
  - CLOCK OUT
  - TOTAL TIME
- The other arrays should be the data of the punches.

For achieving this way of formatting, I created an array variable `csv_data = []`. I retrieved all the punches from database, and once all the data is successfully retrieved in my react component JobDetail, I loop to all of the punches and used the following method to push the punches info as needed:

```
job_detail.punches.map((punch, index) => {
  let arr = []
  arr.push(punch.id);
```

```

        arr.push(moment(punch.punched_in).calendar());
        arr.push(punch.punched_out ? moment(punch.punched_out).calendar() : 'Active as of ' + moment().calendar());
        arr.push(punch.punched_out
        ? moment(punch.punched_out).diff(moment(punch.punched_in), 'hours', true).toFixed(2)
        : moment().diff(moment(punch.punched_in), 'hours', true).toFixed(2)
        )
        csv_data.push(arr)
    })

    csv_data.push([]);
    csv_data.push(["Total hours:", "", "", total_hours]);

```

To calculate the total time spent on each punch, I simply calculate the time in hours between the clock in time and the clock out time. To achieve this in a correct way, I used the library `moment.js`, which is a wonderful and helpful library when it comes to manipulating dates and times. In my case, I use this library in the following way:

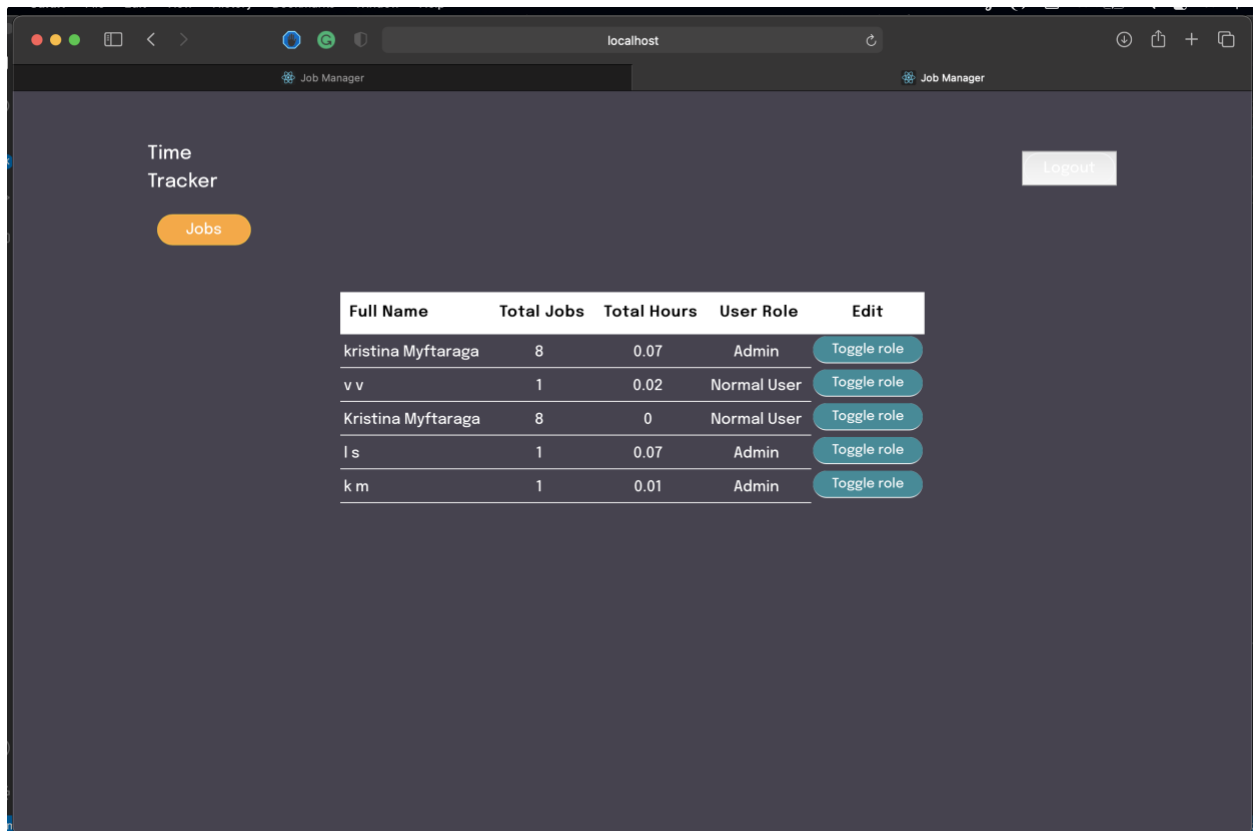
```
moment(punch.punched_out).diff(moment(punch.punched_in)
```

The keyword “diff”, returns the difference between the datetimes `punched_out` and `punched_in`. The “beauty” formatting of the dates, when showing to excel are done the following way: `moment(punch.punched_in).calendar`. The `.calendar` keyword of `moment.js` library takes the datetime generated by `moment()` command and transforms it to a calendar represented datetime like the following: Today at 11:09 AM.

This is very helpful when it comes for being user-friendly.

## ADMIN FUNCTIONALLITY

One new main functionality added for the second phase was the admin functionality. To make distinction between normal users and admin users in the database, I added a field 'is\_admin' to the user model. That allows me to easier restrict the admin pages from normal users and to give permissions. As an admin functionality, I added a new page in the website that lists all the users in our database:



The screenshot shows a web application interface with a dark theme. At the top, there's a navigation bar with 'Job Manager' on the left and a 'Logout' button on the right. Below the navigation bar, the main content area has 'Time Tracker' on the left and an orange 'Jobs' button. In the center, there's a table with the following data:

Full Name	Total Jobs	Total Hours	User Role	Edit
kristina Myftaraga	8	0.07	Admin	<button>Toggle role</button>
v v	1	0.02	Normal User	<button>Toggle role</button>
Kristina Myftaraga	8	0	Normal User	<button>Toggle role</button>
I s	1	0.07	Admin	<button>Toggle role</button>
k m	1	0.01	Admin	<button>Toggle role</button>

As it is visible in the photo, there is a list of users and some info about them. First column shows their full name, the second column the number of jobs added, the third column the total number of hours worked in all their jobs, and the fourth column shows the user role for each person. This feature can be very helpful for platform admins to see how many users are using the platforms and how they interact with the platform.

For calculating the number of total hours for each users, I used a helper method in the backend, which takes a user id and returns the needed result. The code looks like the following and I will explain it in more detail:

```

const get_total_hours = async (id) => {
  let hours = 0
  // Get all punches and calculate sum of hours
  const punches = await Punch.find({ user:id })
  console.log('punches',punches.length)
  punches.forEach(punch => {
    const curr_hours = Number(punch.updatedAt
    ? moment(punch.updatedAt).diff(moment(punch.createdAt), 'hours', true).toFixed(2)
    : moment().diff(moment(punch.updatedAt), 'hours', true).toFixed(2))
    console.log(moment(punch.updatedAt))
    console.log(moment(punch.createdAt))
    hours += curr_hours
  })
  return hours
}

```

The argument is the id of the user. Then, it queries the database to filter all the punches of this particular user. Keeping a variable hour – variable to return – it loops through all the punches and, like explained in the csv report – I used the moment library to find the time in hours between the clock out and the clock in time. Then, the punch time for each punch is added to the hour’s variable – which is later returned.

This function described above is used for every user that is returned with the for the admin page:

```

userRouter.get('/api/users', async (request, response) => {
  const allUsers = await User.find({}).populate('jobs',{ title:1 })

  const users = []

  for(let i =0; i< allUsers.length; i++){
    let curr_user = allUsers[i]

    const total_hours = await get_total_hours(curr_user.id)

    users.push({
      ...curr_user._doc,
      total_hours
    })
  }

  response.status(200).json(users)
})

```

When users are registered, every new user will have the is\_admin field equal to false. I created one admin user by changing the is\_admin field manually from the database. Then, I added a new functionality for admins that can change the other user’s role by just clicking a button. As seen from the table with users, there is a button “Toggle role”:

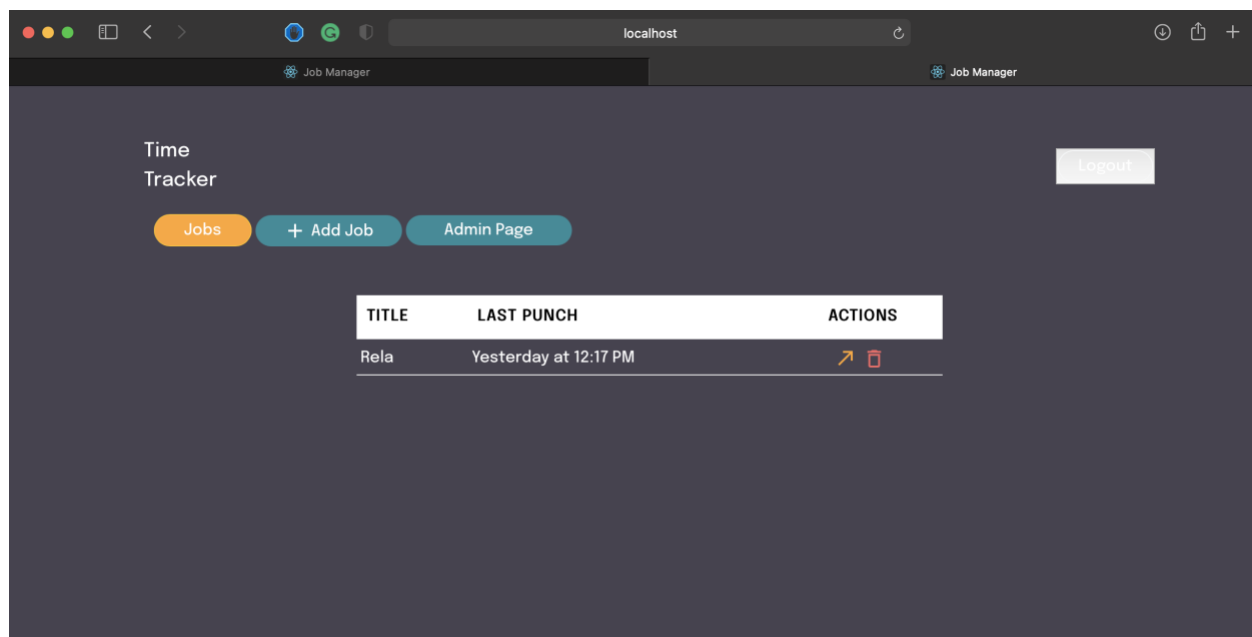


Full Name	Total Jobs	Total Hours	User Role	Edit
kristina Myftaraga	8	0.07	Admin	Toggle role

When an admin user go clicks there, there will be an event handler that will send a request to the backend. This request will be of type PATCH and will just update the user role. If the user is admin, then it's role will be normal user, and if user is normal, its role will be admin.

Here is a test on how it will work with 2 different users taking part.

Initially, the admin user can see the following page when he logs in:

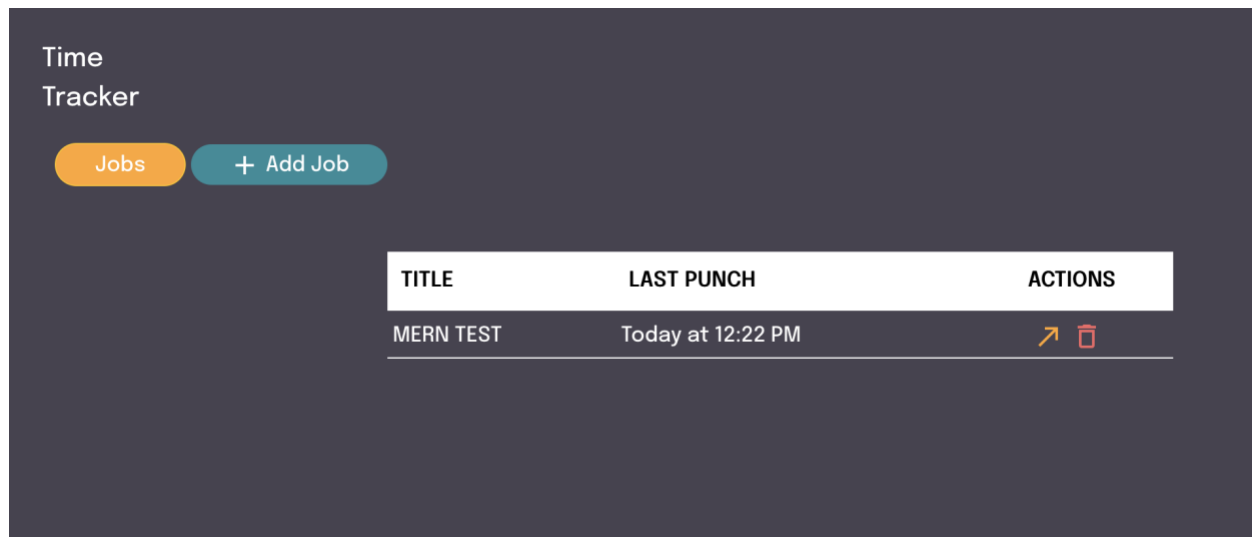


The button named "Admin Page" will lead the user to the list with all the users (attached in earlier figures).

After going to the admin page, the list of users will show and we will take in consideration the user with full name "k m"

k m	1	1.17	Normal User	Toggle role
-----	---	------	-------------	-------------

It is visible that this user is normal user and the following page will show in its landing page:



So, the button Admin page is not visible here. After clicking “Toggle role” button then if the user “k m” log in again, the menu will also include the “Admin page” button. Consequently, user “k m” will now hold the admin privileges for the Time Tracker website.

Code in the backend server for toggling user role:

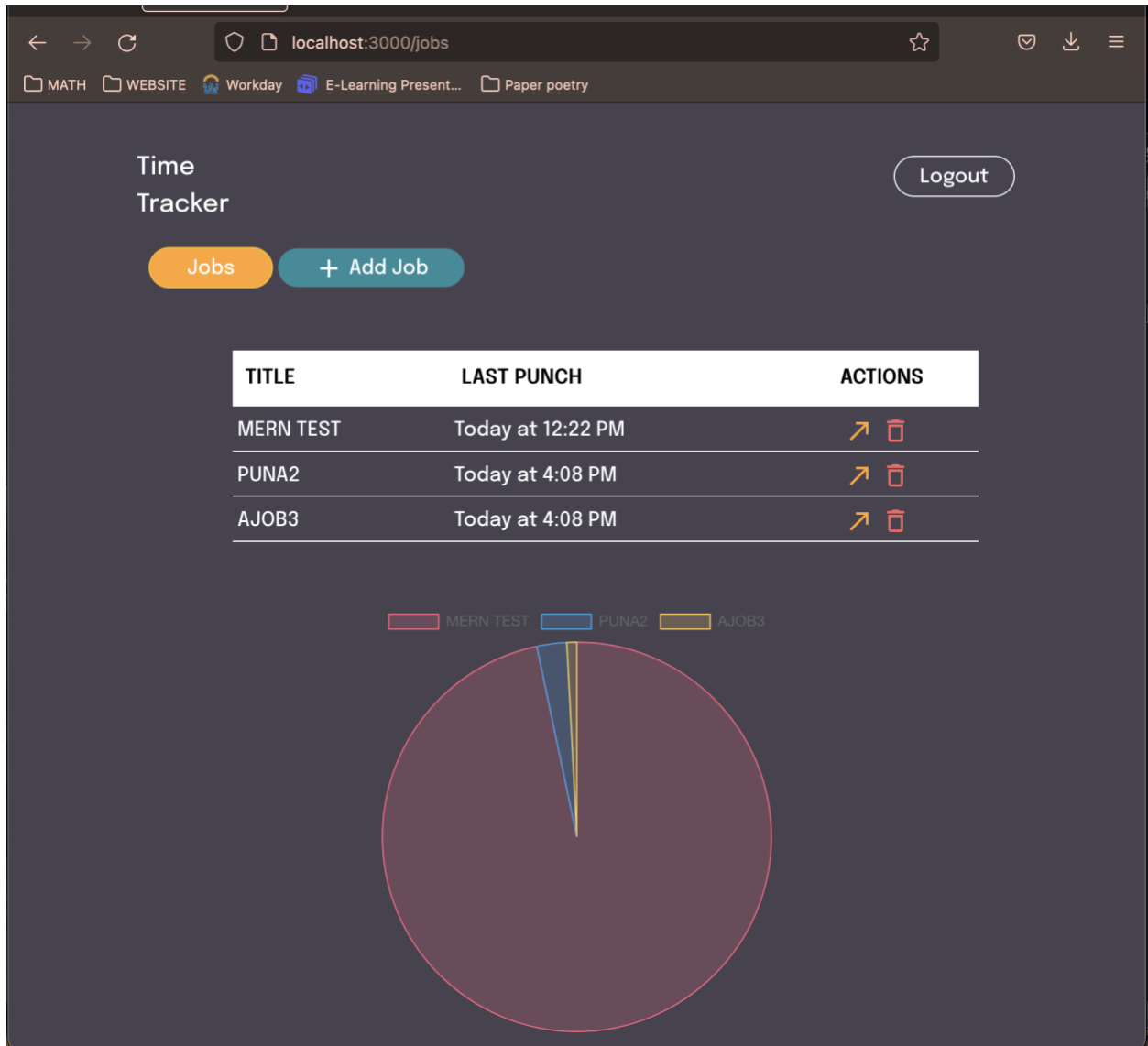
```
userRouter.patch('/api/admin/toggle/:user_id', async (request, response) => {
  const user_id = request.params.user_id
  const user = await User.findById(user_id)
  user.is_admin = !user.is_admin
  await user.save() // saves the user
  response.status(204).json({})
})
```

## VISUALIZATIONS

Charts to visualize some statistics for admin user regarding other users and their worktime:

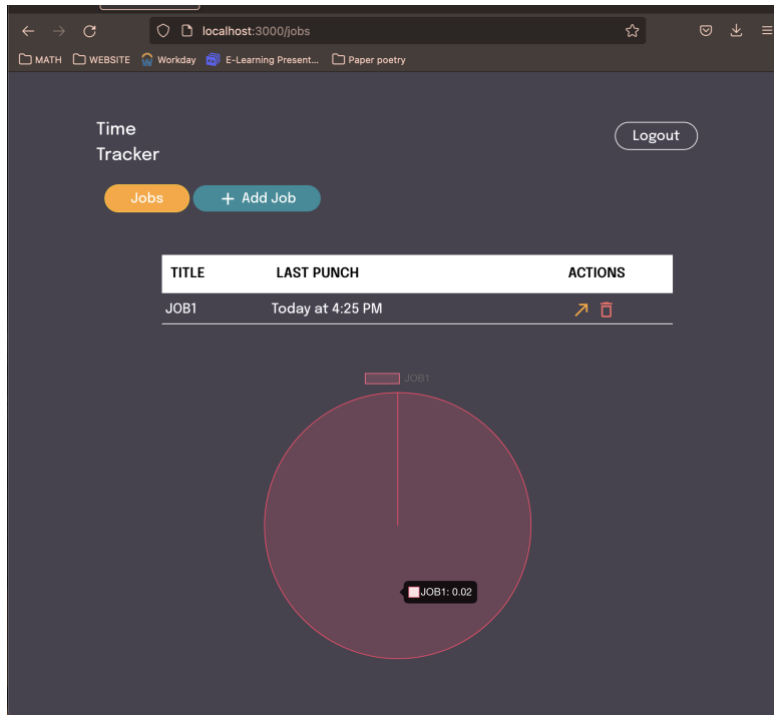
<https://www.chartjs.org/docs/latest/samples/legend/events.html>

Usually, people are interested to know how much time they spent on each job. However, it is not enough to know how much time we spent on each job, rather to see that. For these reasons I have decided to add a visualization like below.

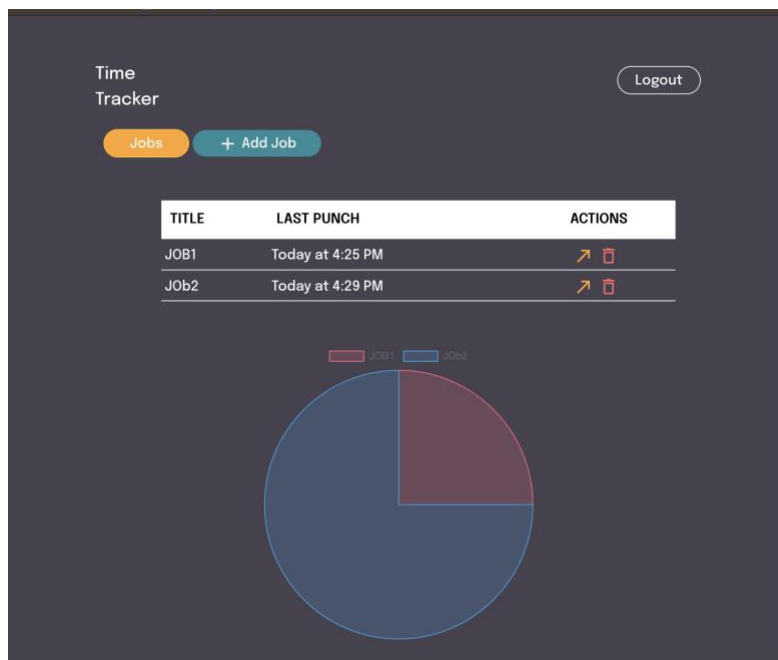


The pie chart in the end of the page will show how much time user spent on each job. In this way they will have a visualization of how much time they need to spend on each task.

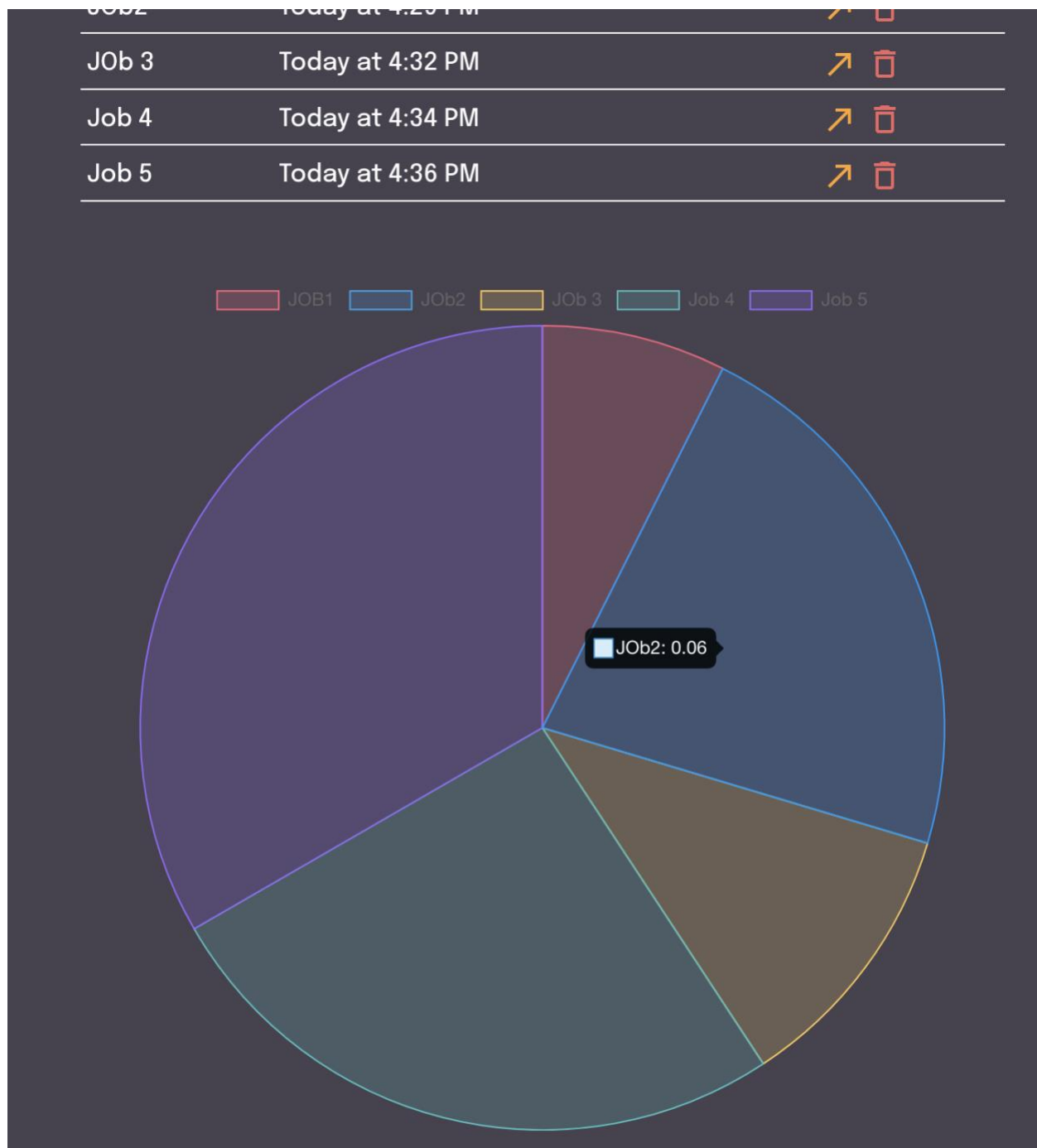
Here is how the chart looks when there is only one job.



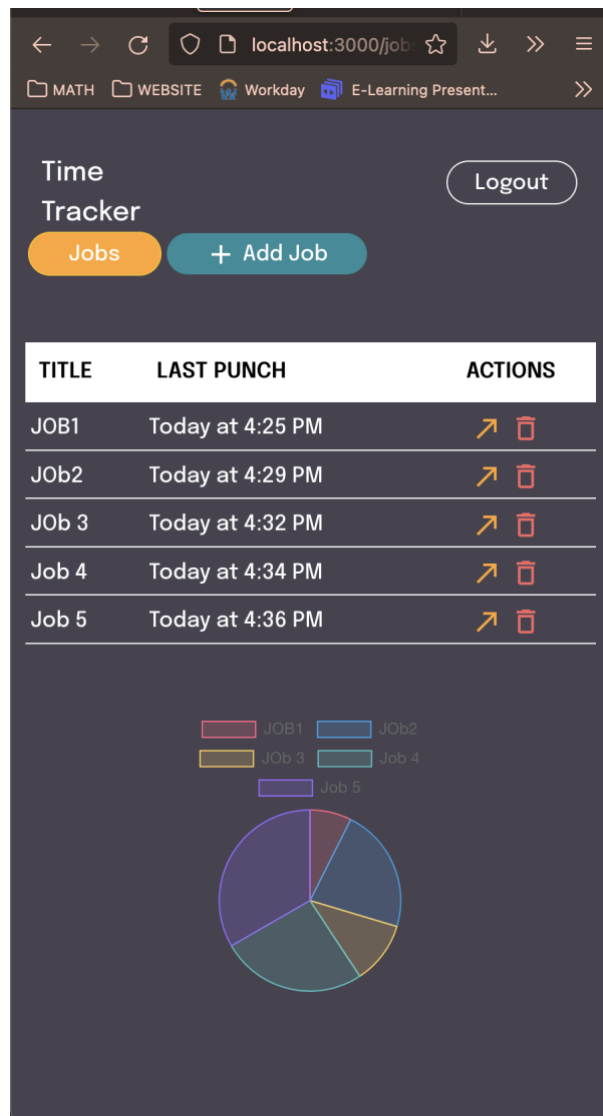
Worth mentioning is that when the user is currently punched in, the chart will not reflect the time spent for that punch on that specific job. Immediately after the user will punch out the latest session will reflect on the pie chart.



And the chart below shows how it looks when there are more jobs added.



The charts provide some very user-friendly effects like when hovering on some part of the chart, it will show the name of that job and the total hours spent there. When we minimize the screen, the chart is also responsive. Following screenshot is how the jobs page looks like when opened in a small screen (mobile view):



## Implementation of Charts.

For the pie chart implementation, I made use of the Chart.js library and its API. I was researching on ways how to do these kinds of visualizations and Chart.js seemed very convenient as it was free, open source and very popular amongst developers. It also provided an integration to React.js – which made the integration to my project even easier. Here is how I set up this library for my project:

I installed the package react-chartjs-2 via npm in my frontend application. After that, I imported the necessary components:

```
import { Chart as ChartJS, ArcElement, Tooltip, Legend } from 'chart.js';
import { Pie } from 'react-chartjs-2';

ChartJS.register(ArcElement, Tooltip, Legend);
```

The ChartJS.register(...) makes sure that our visualization component will render with all the necessary sub-components as Legend and Tooltip. The Pie component imported from react-chartjs-2 is where the data about the jobs will be visualized – it contains the PieChart.

The config object looks as the following:

```
data = {
  labels: jobs_data.map(j=>j.title),
  datasets: [
    {
      label: '# hours for each job',
      data: jobs_data.map(j=>j.total_hours),
      backgroundColor: [
        'rgba(255, 99, 132, 0.2)',
        'rgba(54, 162, 235, 0.2)',
        'rgba(255, 206, 86, 0.2)',
        'rgba(75, 192, 192, 0.2)',
        'rgba(153, 102, 255, 0.2)',
        'rgba(255, 159, 64, 0.2)',
      ],
      borderColor: [
        'rgba(255, 99, 132, 1)',
        'rgba(54, 162, 235, 1)',
        'rgba(255, 206, 86, 1)',
        'rgba(75, 192, 192, 1)',
        'rgba(153, 102, 255, 1)',
        'rgba(255, 159, 64, 1)',
      ],
      borderWidth: 1,
    },
  ],
};
```

I included a variety of colors for each Pie piece just to make it more beautiful and more recognizable amongst the jobs – both background colors and border colors.

As for the datasets, the object expects to provide a list **label** that will contain all the labels for the pie components – in my case, it will show the title of the jobs.

Secondly, the most important thing that the data object expects is the **data** array. This is expected to be an array of numbers – each number will determine the size of each pie component, and each number is connected with the labels with the same index. For example, the label with index 2 (label[2]) will correspond to the number with index 2 at the data array – data[2]. It will mean that the job and index 2 will have data[2] hours that will be visualized in the pie chart.

To have all this information ready in this frontend component, I prepared it from the backend side. When the server sends the list with all the jobs for specific users, it will also include the number of hours for each job.

This can be achieved from summoning the time in hours of each punch corresponding to the jobs. I implemented a helper method in the jobs controller (in the backend) that, given as argument a job id, will return the sum of all the hours connected with that job. This method is very similar to what I used to calculate the number of hours worked for each user:

```
const get_total_hours = async (id) => {  
  let hours = 0  
  // Get all punches and calculate sum of hours  
  const punches = await Punch.find({ job:id })  
  
  punches.forEach(punch => {  
    const curr_hours = Number(punch.updatedAt  
      ? moment(punch.updatedAt).diff(moment(punch.createdAt), 'hours', true).toFixed(2)  
      : moment().diff(moment(punch.updatedAt), 'hours', true).toFixed(2))  
  
    hours += curr_hours  
  })  
  
  return hours  
}
```

Place where this method is called:

```
jobRouter.get('/api/jobs', async (request, response) => {  
  const curr_user = await get_user_from_request(request)  
  if (!curr_user) return response.status(401).json({ error:'missing or invalid token' })  
  const allJobs = await Job.find({ user: curr_user })  
  const jobs = []  
  for(let i=0; i<allJobs.length; i++){  
    const total_hours = await get_total_hours(allJobs[i]);  
    jobs.push({
```



```

        ...allJobs[i]._doc,
        total_hours
      })
    }
    response.status(200).json(jobs)
  })
}

```

So, now when the frontend queries all the jobs connected with specific users, here is how the response payload will look like.

Array with elements in the following format (a sample response element is shown below):

```

_id: "62654f86dae3472127dd507e"
createdAt: "2022-04-24T13:24:22.915Z"
punched_in: false
punches: Array [ "62654f8edae3472127dd508b" ]
title: "JOB1"
total_hours: 0.02
updatedAt: "2022-04-24T13:25:58.223Z"
user: "622a5cf704c262abd232f24f"

```

So, having a list with these objects, it is easy now to extract what we want.

For extracting the titles of each job (to fill the **labels** array), the following map works successfully:

```
jobs_data.map(j=>j.title),
```

For retrieving the total hours worked, the following map is used:

```
jobs_data.map(j=>j.total_hours),
```

After all these manipulations I did with the data, now what is left is just to use the react component prepared by Chart.js and render the Pie Chart:

```
{data && <Pie data={data} />}
```

I used the && to make sure that the data array is defined. This is a very useful check that ensure that what we pass to the Pie component is defined and hence skip any possible errors.

#### *Possible future work in this application*

In the current state, the application supports the following features:

- User authentication
- User roles (admin/normal users)
- All users insert/delete some job
- All users can punch in and punch out to their added jobs
- All users can see the list of their jobs and the punches of each job
- All users can see detailed information about their punches by exporting the data via CSV
- All users can see smooth and user-friendly visualizations of their punches for all the jobs they have inserted.
- Admin users can see a list of all the users registered and additional info about them (name, number of jobs, total hours worked)
- Admin users have the option of toggling other user roles withing the platform (from admin to normal user or from normal user to admin)

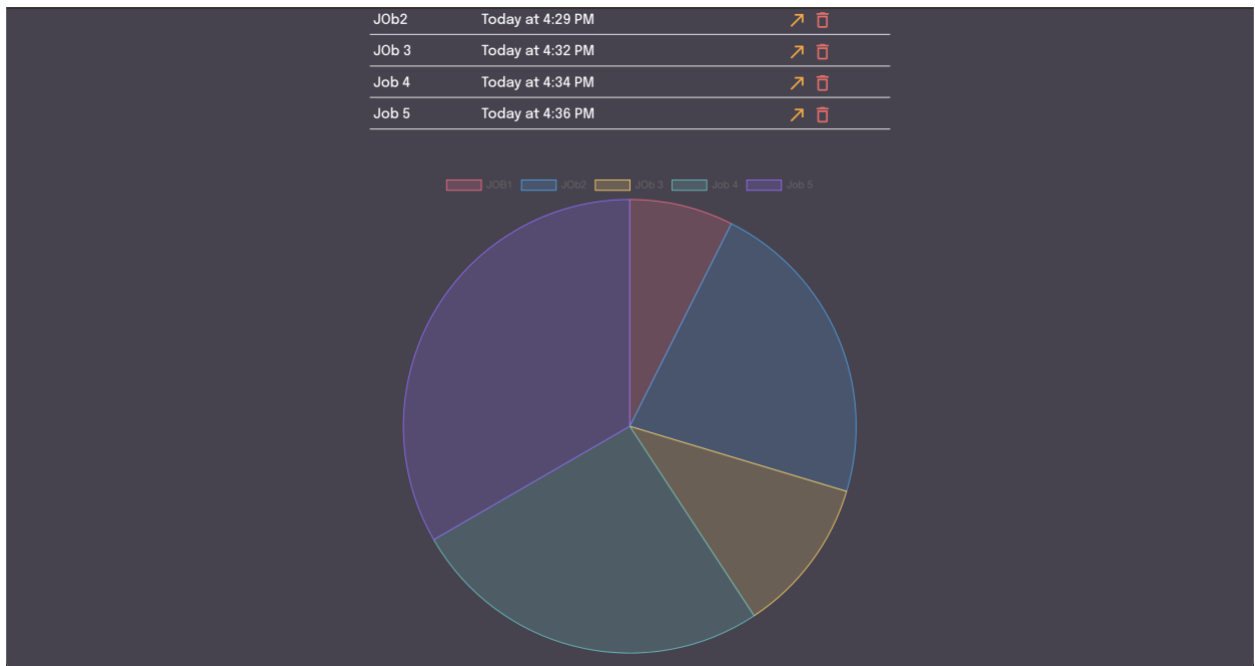
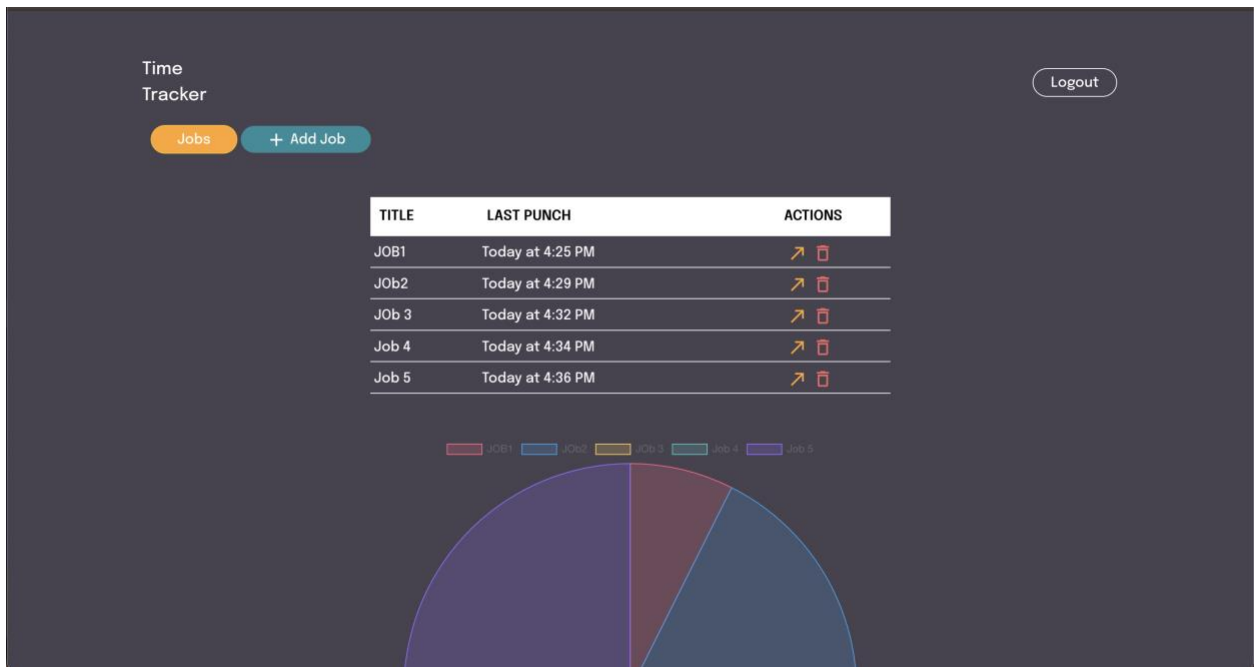
Some possible and very useful features that I will work in the future can be the following:

- Admin users to see visualization about details for other users' jobs and punches
- Admin users to export a list of all the users and additional info about them
- All users to receive notifications via email as reminders to punch in or punch out for specific days.
- All users to also include the hour rate and thus create reports and visualizations with money.

There are a lot of new potential features that come into my mind when it comes to this project. I learned a lot building this application and I am very happy with the end result. The development strategies I used are very up to date when it comes to technologies used. The fact

that I have a separate backend and separate frontend makes it even better, because it also makes it easier to integrate a mobile app consuming the data from the backend.

Here are some screenshots that demonstrate the final application at the current state:



Time Tracker

Jobs

Logout

JOB1

(1 clock ins)

PUNCH IN

#	CLOCK IN	CLOCK OUT	TOTAL HOURS
1	Today at 4:24 PM	Today at 4:25 PM	0.02
TOTAL HOURS:			0.02

Export as CSV

